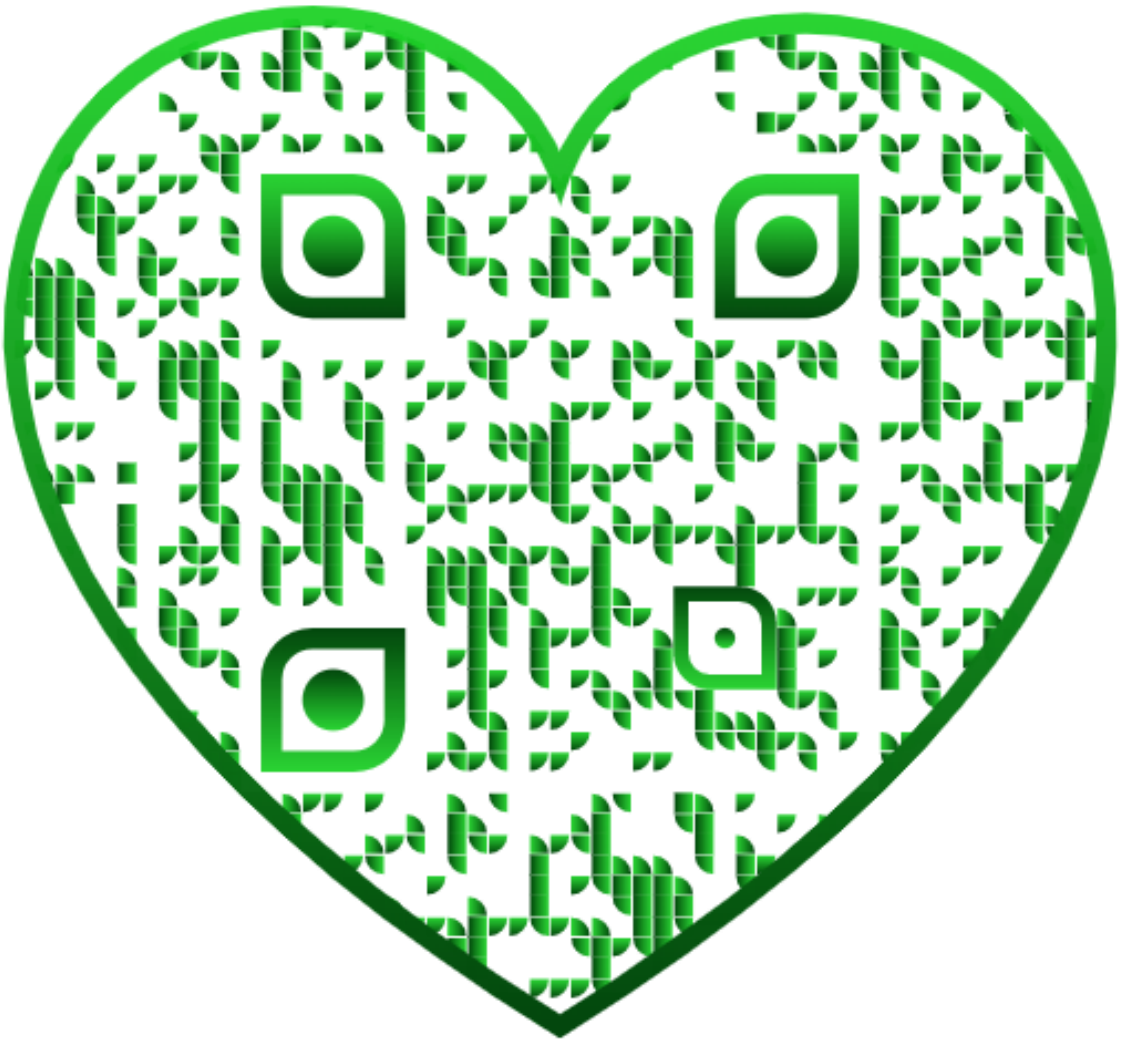
DÉVELOPPEMENT DE JEUX **AVEC UNITY 5**

L'ESSENTIEL POUR LE DÉVELOPPEMENT PC/WEB ET MOBILE

AUTEUR : MARC-ANDRÉ LAROUCHE



unity

Préface

Pour biens des gens, développer un jeu vidéo peut sembler exceptionnellement ardu. La combinaison de codes, d'art, de sons et de technologies qui s'harmonisent pour donner naissance à une source inépuisable de divertissements est un mystère qui, à priori, semble requérir une équipe de passionnés qualifiés. La réalité est beaucoup plus simple quand on connaît les différents outils de développement qui sont aujourd'hui disponibles sur le net. Unity® fait partie de ces outils de développement qui sont gratuits, faciles à utiliser et qui ouvrent la porte aux développeurs indépendants.

Ce livre a pour objectif principal de vous apprendre la façon de débuter dans le logiciel, à partir de différents projets que vous pourrez jouer à la fin des exercices. La complexité des exercices augmentera au cours des différents chapitres. Nous aborderons également plusieurs techniques de créations utilisées dans l'industrie du jeu vidéo. Avec un peu d'expérience en programmation et les notions couvertes dans ce livre, vous devriez avoir tous les outils nécessaires pour réaliser vos propres jeux.

De plus, ce manuel a pour objectif d'être le plus actuel possible afin de ne pas décevoir ceux et celles qui l'ont acheté pour une mise à niveau des connaissances avec la dernière version d'Unity, tout en étant accessible aux nouveaux utilisateurs.

Ce dont vous aurez besoin pour une utilisation optimale

• Vous devrez posséder un ordinateur personnel qui répond aux exigences minimales d'Unity équipé du système d'exploitation Windows ou OS X.
• La version 5.0 ou supérieure du logiciel http://Unity.com/Unity/download
• D'un compte utilisateur Unity pour acheter ou télécharger différentes composantes sur le « *Assets store* » https://accounts.Unity.com/sign-up.
• Vous devez également télécharger les ressources créées spécifiquement pour les exercices de ce livre à l'adresse : http://1drv.ms/1FyzdlO

D'autres ressources vous seront signalées au fur et mesure que progresserez dans les exercices.

À qui s'adresse ce livre

Niveau : Débutant – Intermédiaire

Ce livre s'adresse à tous les amateurs, peu importe le niveau de connaissances en développement de jeux. Il s'adresse également aux développeurs qui migrent d'un autre éditeur comme UDK, Flash, Virtools et Game Maker Pro.

Matière couverte dans le livre

Chapitre 1, *Un vrai coffre au trésor !*

Il vous introduit à Unity, cet engin de jeu qui vous permet de créer et déployer vos jeux sur une panoplie de plateformes, dont notamment le Web, les PC (*Windows, OSX et Linux*), les mobiles (*iOS, Android, Windows, Blackberry*) ainsi que les consoles de jeux (*Xbox 360/One, PlayStation 3/4/Vita/Mobile et Wii U*) et plus. Dans ce chapitre, vous aurez la chance de jouer à différents exemples de jeux qui ont été créés à partir d'Unity, de télécharger et d'installer votre copie du logiciel ainsi que quelques démos fournis par la compagnie.

Chapitre 2, *Jeu #1 : Rebonds*

Dans ce chapitre, vous allez créer votre premier projet avec Unity. On y explore l'environnement de travail ainsi que les composantes de base comme les ***colliders***, le ***rigidBody*** et les ***physic Materials***. Vous apprendrez à créer un jeu qui consistera à faire rebondir une balle autant de fois que possible sans la laisser tomber, tout en utilisant la physique native incluse dans l'engin.

Chapitre 3, *Premiers pas en programmation JavaScript*.

Avant de se lancer tête première dans le code de l'exercice **Rebonds**, nous allons découvrir les notions de base du langage JavaScript ainsi que les concepts essentiels que tous les programmeurs devraient connaître : du pseudo-code jusqu'à l'héritage.

Chapitre 4, *Programmation de Rebonds : partie 2*

Elle poursuit l'exercice du chapitre 3 en y ajoutant quelques scripts qui rendent le jeu interactif. À partir de codes qui vous seront expliqués, vous allez permettre à une plateforme de bouger avec le déplacement de la souris ou les touches du clavier. À la fin de ce chapitre, vous aurez un premier jeu qui saura vous captiver durant des heures.

Chapitre 5, *Les chronomètres*

Nous allons réaliser trois types de chronomètres pour vos jeux futurs: une version numérique, une version en barre ainsi qu'une autre en pointe de tarte. Toutes ces versions utilisent le même code de base, ce qui facilitera la réutilisation de ces composantes.

Chapitre 6, *Jeu #2 : Match-deux*

Il vous introduit à une notion indispensable pour la création de vos projets : les interfaces utilisateurs (IU ou UI en anglais). Vous pourrez observer la création et la disposition de boutons, champs de textes, logos et menus, en utilisant les nouveaux outils que l'on retrouve à partir de la version 4.6 et 5 de l'éditeur. Ensuite, nous allons y ajouter du code et des animations pour enrichir votre deuxième jeu. Nous limiterons le nombre d'essais qu'un joueur peut effectuer et mélangerons l'ordre des cartes afin de rendre le jeu exclusif à chaque partie. À la fin de ce chapitre, vous devriez être en mesure de créer vos propres interfaces et menus pour vos projets personnels.

Chapitre 7, *L'éditeur de terrain*

Nous y traiterons la manière de créer une scène afin de pouvoir se promener dans l'environnement comme un jeu de tir à la première personne (ou FPS). Dans ce chapitre, nous traiterons des outils

d'édition de terrain, de l'éditeur d'arbres, des *skyboxes*, du système d'éclairage global et des effets de lentilles (*Image Effect*).

Chapitre 8, *L'éditeur de terrain : partie 2*

Dans cette section, nous ferons le raffinement du terrain en y ajoutant des particules, des animations et un radar pour guider le joueur.

Chapitre 9, *L'éditeur de terrain : partie 3*

Nous ajouterons une interface utilisateur et des effets spéciaux quand on se retrouve sous l'eau, dans le cœur du volcan et lorsqu'il y a des blessures. Nous verrons également comment créer un menu de pause.

Chapitre 10, *Jeu #3 : Incendie*

Nous ferons un petit jeu simple où le joueur doit attraper des vases antiques. Nous utiliserons le système de particules afin de créer des effets de fumée, d'éclats et d'autres effets spéciaux qui nécessitent des particules animées. Nous aborderons également la théorie touchant le système d'animation ***Mecanim*** afin d'en déclencher pour notre jeu, selon les actions du joueur.

Chapitre 11, *Incendie : partie 2*

Dans cette section, nous complèterons le jeu du chapitre précédent. On y apprendra comment réutiliser les scripts de différents objets et comment l'utilisation de « *prefabs* » peut nous permettre de changer plusieurs instances d'un même objet, en un clic. Pour terminer, on ajoutera des éléments sonores afin de compléter l'ambiance du jeu.

Chapitre 12, *Le développement mobile*

Dans ce chapitre, nous examinerons la plus grande force d'Unity et la création de jeux portables. Nous traiterons de l'installation et configuration d'*Android SDK* et *Apple XCode SDK* pour la création d'application, la configuration et l'utilisation « *d'Unity remote* » sur votre appareil, pour tester vos jeux directement dans l'éditeur avec les entrées de votre téléphone. Nous complèterons ce chapitre avec le développement pour *Apple iOS* avec *XCode*.

Les conventions utilisées dans ce livre

Dans ce livre, vous retrouverez fréquemment différents formats d'en-tête.

Titre de section

Les sections importantes utilisent un en-tête en caractères gras et italiques et sont indexées dans la table des matières :

« Section ABC »

Nouvel atelier

Afin de vous donner des directives claires, vous retrouverez cet en-tête :

Atelier pratique

Les différentes sous-catégories d'un exercice sont souvent accompagnées d'un en-tête comme ci-dessous :

« Sous-catégorie ABC »

Fichiers requis (téléchargés)

Les fichiers requis pour les exercices sont accompagnés de cet icône :

Listes et puces

Les instructions sont numérotées comme ceci :

1. Première étape

1.1. Étape secondaire

Groupe
1.1.1. ***Détails*** : valeur
1.1.2. ***Détails*** **:** valeur

Les listes d'éléments utilisent des puces et des **textes en caractères gras** :

- **Élément 1**
- **Élément 2**
- **Élément 3**

Description

Les instructions ont souvent besoin d'explications, c'est pourquoi elles sont souvent accompagnées par :

En détails…

Vous y retrouverez les explications sur les tâches et instructions que vous venez de compléter. Les (**parenthèses en caractères gras**) contiennent les mots-clés qui font référence au code.

Scripts

Un nouveau bloc de codes sera illustré comme ceci :

```
function Start(){
      transform.position = Vector3 (0,0,0);
}
```

Le code écrit en caractères **gras** sert à attirer l'attention sur les lignes de codes **ajoutées** ou ~~**retirées**~~ :

```
function Start(){
      ~~transform.position = Vector3 (0,0,0);~~
      transform.position = transform.parent.position;
      transform.parent = null;
}
```

Terminologie

Les **termes** et les **mots-clés** sont illustrés en caractères **gras** ou *italiques*.

• L'*italique* représente les différents panneaux (views).
• Les caractères ***gras-italiques*** sont pour les différents termes d'Unity (ex. : ***GameObjects, Audio-Sources, Cameras***)
• Le caractère **gras** représente vos ***assets*** de l'exercice. (ex. : **Spaceship model, Main Camera, FP Controller**)
• « Les guillemets » sont pour les boutons (ex. : « Add Component », « Build », « Apply »)
• Le symbole (|) est utilisé comme séparateur d'arborescence des menus.

Par exemple :

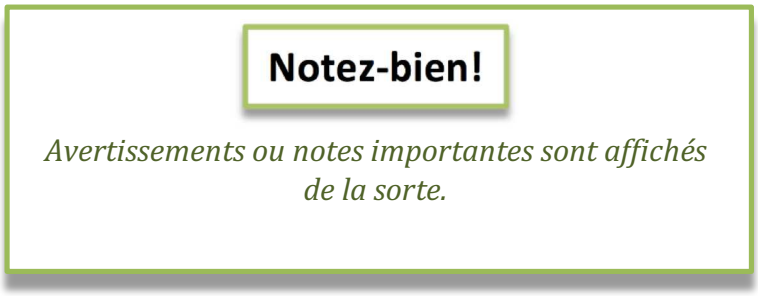
1. À partir du panneau *Project,* glissez le ***prefab*** : **Spaceship model** dans *Hierarchy.*

2. Dans l'*Inspector*, appuyez sur le bouton « Add Component » puis :

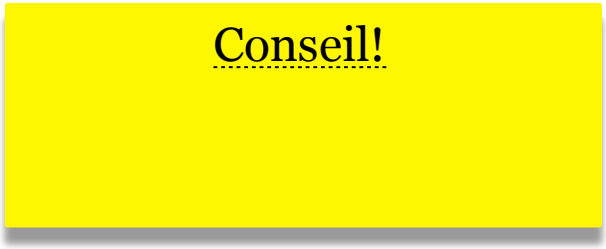
Audio | AudioSource

Les éléments à retenir

Les notes apparaissent comme ceci :



Les conseils pratiques ressemblent à cela :



Liens HTML

Les liens Internet sont soulignés en bleu:

http://www.unity3d.com

Voilà ce qui conclut la liste des conventions pour cet ouvrage.

Remerciements et informations légales

Ce livre est rendu possible grâce à la collaboration des personnes suivantes :

• Rollande Brochu Larouche, pour la correction du français.
• SIL International pour les polices : **Charis** et **Apparatus** qui seront utilisées dans les exercices. Ces polices sont distribuées sous la licence SIL Open Font License. Pour plus de détails, allez à l'adresse Web suivante :

http://scripts.sil.org

PRÉFACE **3**
CE DONT VOUS AUREZ BESOIN POUR UNE UTILISATION OPTIMALE 3
À QUI S'ADRESSE CE LIVRE 3
MATIÈRE COUVERTE DANS LE LIVRE 4
LES CONVENTIONS UTILISÉES DANS CE LIVRE 5
REMERCIEMENTS ET INFORMATIONS LÉGALES 8
CHAPITRE 1 - UN VRAI COFFRE AU TRÉSOR ! **13**
MATIÈRE COUVERTE DANS CE CHAPITRE 13
INTRODUCTION 14
POURQUOI UTILISER UN ENGIN ? 14
MAINTENANT, VOICI UNITY ! 14
À PAS DE BÉBÉ 15
MISE EN ROUTE DU LOGICIEL 17
PROJET D'EXEMPLE 17
CAR 18
CHARACTERTHIRDPERSON ET CHARACTERFIRSTPERSON 19
PARTICULES 19
ROLLERBALL 19
2DCHARACTER 20
AIRCRAFTJET2AXIS 20
AIRCRAFTPROPELLER4AXIS 20
APPRENTISSAGE DE LA TERMINOLOGIE D'UNITY 21
APPRENTISSAGE DE L'INTERFACE 23
PANNEAU (PROJECT BROWSER) 24
PANNEAU (HIERARCHY) 28
LA BARRE D'OUTILS (TOOLBAR) 29
LA FENÊTRE (SCENE VIEW) 29
LA FENÊTRE (GAME VIEW) 33
LE PANNEAU (INSPECTOR) 36
LES FENÊTRES (VIEWS) ADDITIONNELLES 39
PERSONNALISATION DE VOTRE ESPACE DE TRAVAIL 41
LE PROCESSUS DE CRÉATION DES TEXTURES 43
LE PROCESSUS DE CRÉATION DES (GAMEOBJECTS) 45
IMPORTEZ LES PARAMÈTRES DE VOS OBJETS 48
CRÉEZ UN PREFAB 49
CRÉATION DE SCÈNES 50
PUBLIEZ VOS PROJETS (BUILD) 51
CHARGEMENT AU PRÉALABLE 55
LES RACCOURCIS DU LOGICIEL 56
LES PRÉFÉRENCES (PREFERENCES) 58
CHAPITRE 2 - JEU *#1 : REBONDS* **63**
MATIÈRE COUVERTE DANS CE CHAPITRE 63
LE CONCEPT 64
AVANT DE DÉBUTER 65
MISE EN PLACE DES ÉLÉMENTS DU GAMEPLAY 65
L'ENVIRONNEMENT 73
LE RADAR 78
CHAPITRE 3 - PREMIERS PAS EN PROGRAMMATION JAVASCRIPT **83**
MATIÈRE COUVERTE DANS CE CHAPITRE 83

Le pseudo-code 84
La syntaxe 85
Les variables 86
Les opérations 91
Les conditions 95
Les boucles 99
Les fonctions 102
La surcharge de fonctions (surdéfinition) 105
Les fonctions et les variables intégrées 106
Les classes 108
L'héritage 110

CHAPITRE 4 - PROGRAMMATION DE REBONDS : PARTIE 2 113
Matière couverte dans ce chapitre 113
Programmation de Rebonds 114
Déplacer la plateforme 114
Manipuler la balle 117
Utilisation des valeurs aléatoires (Random) 119
Garder la trace du pointage 120
Augmentation de la difficulté 122
Effets de particules 124
Préparer le (build) 127
(Baker les lightmaps) 128
Créer les boutons de l'interface 128
L'habillage sonore 135
Exporter un fichier redistribuable 137
Codes complétés 140

CHAPITRE 5 - LES CHRONOMÈTRES 143
Matière couverte dans ce chapitre 143
Avant de débuter 144
Préparatifs de la scène 144
Les éléments de la scène 145
Chrono en texte 147
JavaScript 148
Chrono en barre 154
Le chrono en pointe de tarte 157
Codes complétés 162

CHAPITRE 6 - JEU *#2 : MATCH-DEUX* 165
Matière couverte dans ce chapitre 165
Avant de débuter 166
Préparation du menu principal 166
Les éléments de la scène 166
Ajout d'un logo dans le menu 172
Préparation de la scène de jeu 176
Codes complétés 193

CHAPITRE 7 - L'ÉDITEUR DE TERRAIN 197
Matière couverte dans ce chapitre 197
Création de terrains 198
Les éléments de la scène 198

CRÉATION DU RELIEF 200
AJOUT D'UN VOLCAN 202
AJOUTS DE TEXTURES 204
SEMONS DES ARBRES 208
L'OUTIL DES BRANCHES 209
DE RETOUR À L'EXERCICE 214
L'OUTIL DE FEUILLAGE 217
DE RETOUR À L'EXERCICE 219
LES PARAMÈTRES DU TERRAIN 222
UNE GOUTTE D'EAU DANS L'OCÉAN 226
AJOUT D'UN CIEL ET DU BROUILLARD DANS LA SCÈNE 228
LIGHTING (SCENE) 230
IMAGE EFFECTS 234

CHAPITRE 8 - LE TERRAIN : PARTIE 2 241
MATIÈRE COUVERTE DANS CE CHAPITRE 241
AVANT DE DÉBUTER 242
RAFFINEMENT DU TERRAIN 242
INTÉRIEUR DU VOLCAN : UTILISATION DES PARTICULES 252
INTÉRIEUR DU VOLCAN : ANIMATION DE LA LAVE 264
CRÉATION D'UN RADAR 268
PRÉPARATION DE LA CARTE 269

CHAPITRE 9 - LE TERRAIN : PARTIE 3 287
MATIÈRE COUVERTE DANS CE CHAPITRE 287
EFFETS SPÉCIAUX ET MENU 288
INTERFACES UTILISATEURS (UI) : AJOUT DE FILTRES DE COULEURS 288
SOUS L'EAU 292
UNE PETITE BAIGNADE DANS LE VOLCAN 294
C'EST DUR SUR LES GENOUX! 299
INTERFACES UTILISATEURS (UI) : AJOUT D'UN MENU DE PAUSE 307
CODES COMPLÉTÉS 324

CHAPITRE 10 - JEU #3: INCENDIE 327
MATIÈRE COUVERTE DANS CE CHAPITRE 327
AVANT DE DÉBUTER 328
LES ÉLÉMENTS DE LA SCÈNE 328
AJOUT DU PERSONNAGE 330
LES EFFETS SPÉCIAUX 339
RÉCOMPENSES ET OBSTACLES 345

CHAPITRE 11 – INCENDIE : PARTIE 2 347
MATIÈRE COUVERTE DANS CE CHAPITRE 347
PROGRAMMATION DU JEU 348
AJOUT D'UN SCRIPT POUR LE DÉPLACEMENT DU PERSONNAGE 348
AJOUT D'UN SOL 353
AJOUT D'UN SCRIPT POUR LES VASES 355
AJOUT D'UN SCRIPT POUR LES DÉBRIS 362
AJOUT D'UNE INTERFACE 364
AJOUT DU GAMEPLAY 368
AJOUT D'UNE DÉFAITE 369
« LIGHT PROBE GROUP » 374

ACTIVATION DU POINTAGE 382
AJOUT D'EXPRESSIONS 385
AJOUT DE SONS 388
AJOUT D'UNE AMBIANCE VISUELLE 395
CODES COMPLÉTÉS 399

CHAPITRE 12 - LE DÉVELOPPEMENT MOBILE 405
MATIÈRE COUVERTE DANS CE CHAPITRE 405
LES ORIGINES 406
CE QU'IL FAUT SAVOIR 409
COMMENT SE DÉMARQUER DE LA COMPÉTITION 410
LE FREEMIUM (FREE2PLAY) 414
COMMENT CONCEVOIR UNE INTERFACE UTILISATEUR 415
LA PARTIE PRATIQUE 417
UNITY REMOTE 417
COMMENT CONFIGURER UNITY REMOTE 417
INSTALLATION SUR IOS 417
INSTALLATION SUR ANDROID 419
UTILISATION DE UNITY REMOTE SUR PC 420
COMMENT COMPILER POUR ANDROID 423
EXPORTER SUR ANDROID DANS UNITY® 425
INSTALLATION DU JEU 430
COMMENT COMPILER POUR APPLE IOS 432
EXPORTER SUR IOS DANS UNITY® 444
INSTALLATION DU JEU 449

GLOSSAIRE 453

Chapitre 1 - Un vrai coffre au trésor !

Matière couverte dans ce chapitre

- **Introduction sur les engins**
- **Présentation du logiciel**
- **Premier contact avec un jeu réalisé sur Unity**
- **Installation du logiciel**
- **Mise en route**
- **Les fondements des outils et menus de l'éditeur**
- **Les raccourcis clavier**
- **La personnalisation du logiciel**

Introduction

Le développement de jeux vidéo a connu un amalgame de changements au cours des dernières décennies. Dans les années 70, une seule personne était impliquée dans la création d'un jeu et ce nombre a augmenté de façon drastique vers la fin des années 90. De nos jours, les titres AAA peuvent employer plusieurs studios dans différents pays pour combler la demande afin d'introduire un jeu sur le marché, après 3 à 5 années de production !

Heureusement, avec l'apparition d'engins de développement abordables et parfois gratuits, qui offrent des solutions pour simplifier le développement et la publication de jeux, l'industrie a connu un engouement pour les jeux indépendants. Aujourd'hui, Microsoft, Sony et Nintendo ont ouvert leurs portes aux individus (Indie) qui veulent publier leurs créations sans devoir investir des millions en recherche et développement.

Pourquoi utiliser un engin ?

Construire un jeu est un processus long et dispendieux, les attentes des joueurs sont élevées et les coûts de développement d'un engin interne demandent un gros investissement et causent parfois des frustrations chez les développeurs.

De plus, les engins offrent des outils de développements complets, ils réduisent considérablement le temps de production et d'apprentissage des outils. Comme ils sont souvent bien documentés, c'est plus facile de recruter un spécialiste déjà formé ou de trouver une solution rapide par le biais d'une communauté de développeurs.

Étant donné qu'un engin contient plusieurs outils (animation, programmation, création de particules, etc.), les différents départements pour le montage du jeu peuvent travailler en parallèle et réduire le va-et-vient lors de la production entre ceux-ci.

Maintenant, voici Unity !

Unity® fit sa première apparition publique au Apple's WWDC en 2005, après quatre années de développement. Pendant plusieurs années, Unity est resté dans l'ombre de son compétiteur Virtools. Celui-ci avait pignon sur rue dans plusieurs boîtes de jeux vidéo qui l'utilisaient pour le prototypage. Finalement, voulant innover, Unity a décidé d'amener son engin sur des plateformes mobiles. Ceci a permis de gagner en popularité et de rivaliser avec les engins haut de gamme comme Unreal Engine (UDK).

L'équipe derrière Unity est constamment en production afin d'ajouter de nouvelles fonctionnalités, composantes et plateformes. Au moment où cet ouvrage est rédigé, Unity peut créer des jeux pour iPhone, iPod touch, iPad, appareils Android, téléphones Windows, Blackberry, Xbox 360/One, PlayStation 3/4/Vita/Mobile et Nintendo Wii/Wii U. Certaines de ces plateformes peuvent être exportées gratuitement et d'autres requièrent des licences.

Avec l'acquisition de connaissances contenues dans ce manuel, vous serez en mesure d'approfondir celles-ci et créer des projets de plus en plus complexes. Le truc, c'est de faire des projets auxquels vous pourrez compléter, pour ensuite y ajouter de nouvelles composantes qui rendront ceux-ci encore plus intéressants et captivants.

Atelier pratique

À pas de bébé

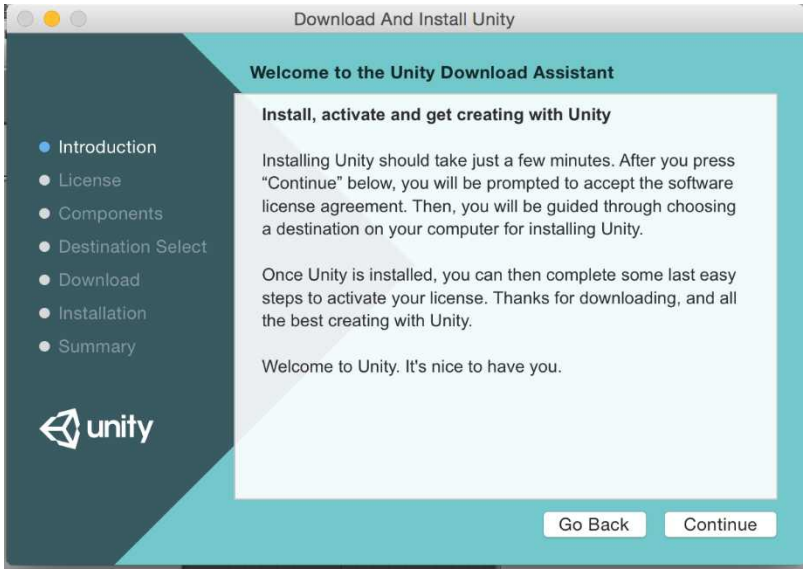
Lorsqu'on se lance dans le développement de jeux, il est important et primordial d'être réaliste par rapport à la qualité du travail demandé et ne pas sous-estimer la quantité de travail nécessaire pour mener à terme un projet. Même avec l'expérience, il est facile de se laisser emporter par un projet trop complexe.

C'est pourquoi, il est préférable de débuter avec de petits projets et d'augmenter graduellement le niveau de difficulté au fur et à mesure de votre expérience. Lors du parcours de ce livre, vous devriez faire quelques petits jeux qui évolueront pour devenir plus complexes au cours des chapitres.

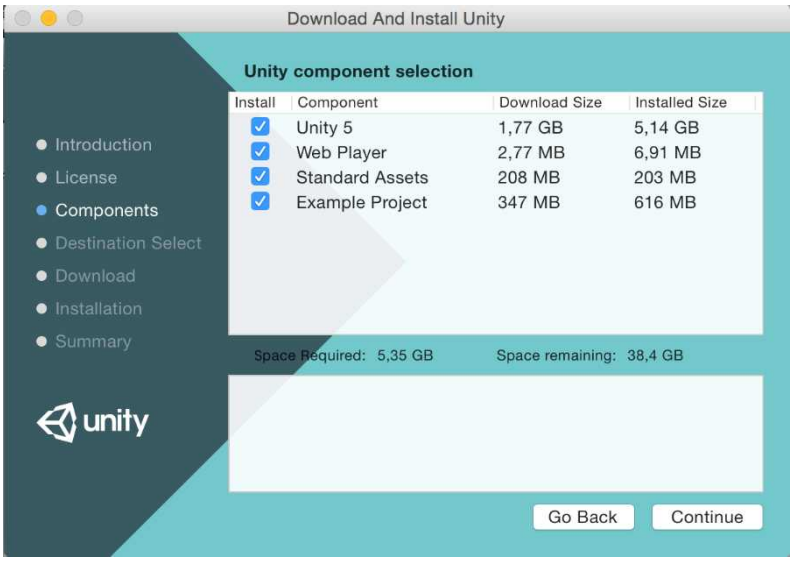
1. Allez sur le site http://unity3d.com/get-unity/download?ref=personal

2. Cliquez sur le bouton pour télécharger la dernière version du logiciel.

3. Suivez les instructions d'installation, étape par étape.



À l'étape : **Components**, assurez-vous que **Standard Assets** et **Example Project** soient présents dans votre sélection.

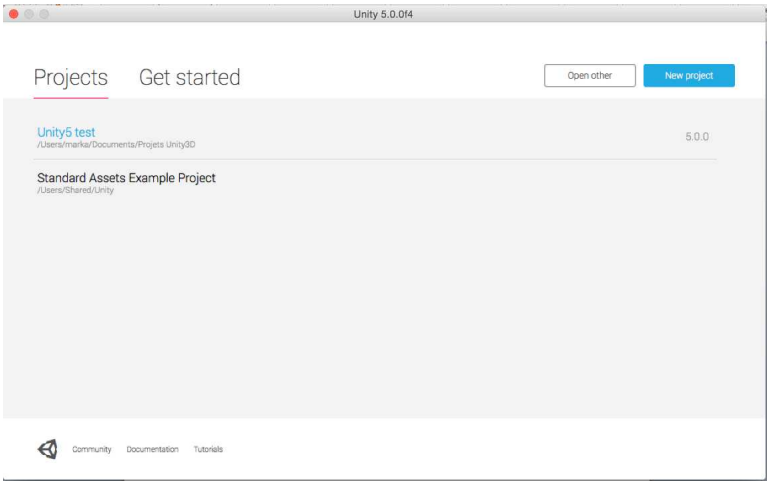


4. La procédure est similaire sur Windows et relativement facile à installer.

Vous avez maintenant la dernière version du logiciel, vous allez donc faire le tour des différents menus et outils afin de vous familiariser avec ses conventions et nomenclatures.

Mise en route du logiciel

5. Lancez le logiciel, vous serez accueillis par cet écran :



La version 5 du logiciel Unity utilise une nouvelle fenêtre d'accueil avec plus d'options pour les débutants.

Si vous utilisez pour la première fois ce logiciel, vous êtes invités à regarder la vidéo qui se trouve dans l'onglet « Get started » (en anglais uniquement).

Dans le bas de la fenêtre, il y a trois liens hypertextes qui vous amènent directement sur les différents sites d'aide pour les débutants : ***Community,*** où les gens échangent leurs problèmes et solutions, ***Documentation,*** pour combler tous vos besoins dans le logiciel et finalement, ***Tutorial,*** afin de vous entraîner avec les différents outils.

Puisque tous les documents et tutoriels sont en anglais et espagnol, ce manuel en français viendra à la rescousse de ceux qui ne parlent pas la langue de Shakespeare.

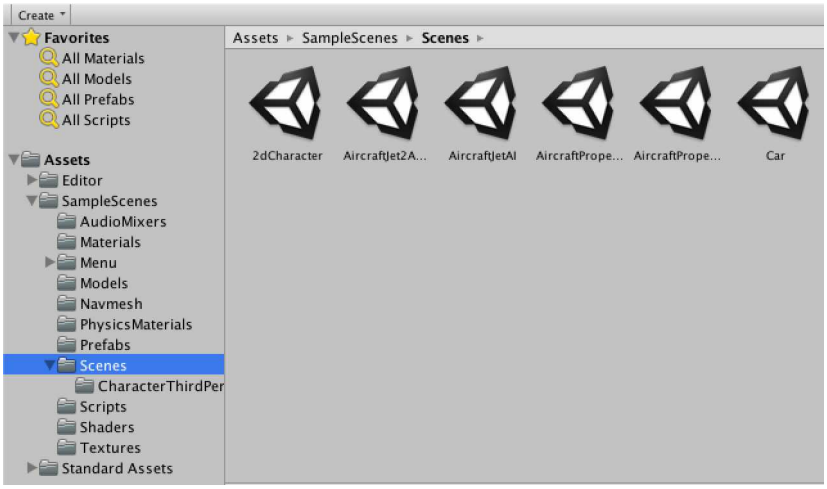
Projet d'exemple

6. Appuyez sur le projet : **Standard Assets Example Project.**

Lorsque le projet est ouvert, localisez le panneau nommé : *Project,* il devrait se trouver dans le bas de l'interface, par défaut.

7. Naviguez jusqu'au répertoire :

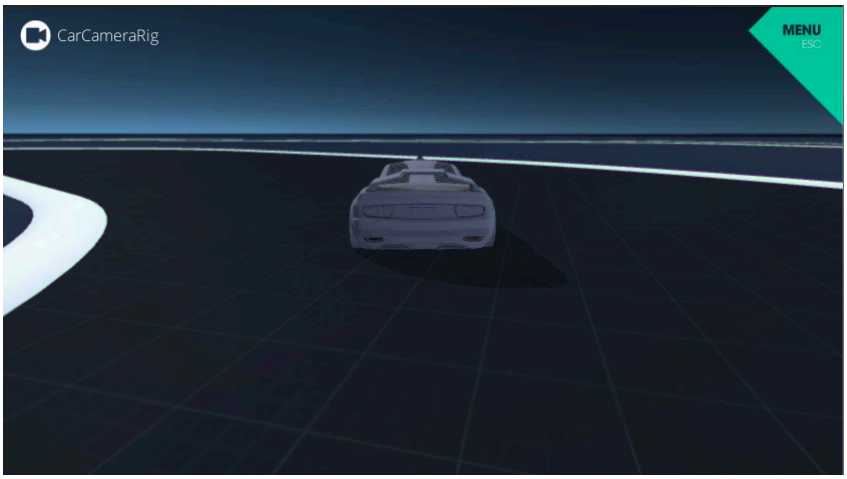
SampleScenes | Scenes



Celui-ci vous donne accès à plusieurs exemples de scènes qui indiquent différentes possibilités qu'Unity peut effectuer.

Essayez quelques scènes afin de vous familiariser avec les exemples.

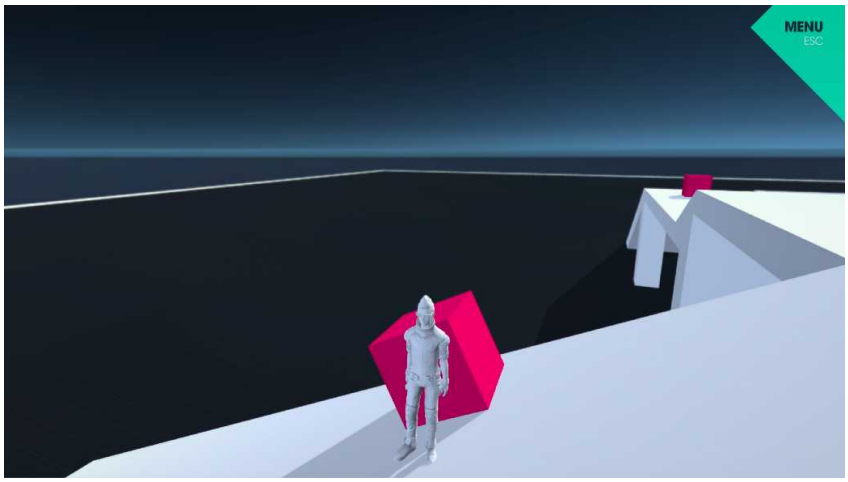
Car



Cette scène contient une piste de course, une voiture et plusieurs obstacles activés par la physique. Vous pouvez vous inspirer de cette scène ainsi que de la scène **CarAI-WaypointBased** pour réaliser un jeu de course, en très peu de temps.

↘ ***Car*** *et* ***CarAIWaypointBased***

CharacterThirdPerson et CharacterFirstPerson



Présentement, nous voyons comment le système d'animation d'Unity (***mecanim***) peut mixer différentes animations selon divers paramètres. Le personnage peut interagir avec son environnement et les obstacles distincts.

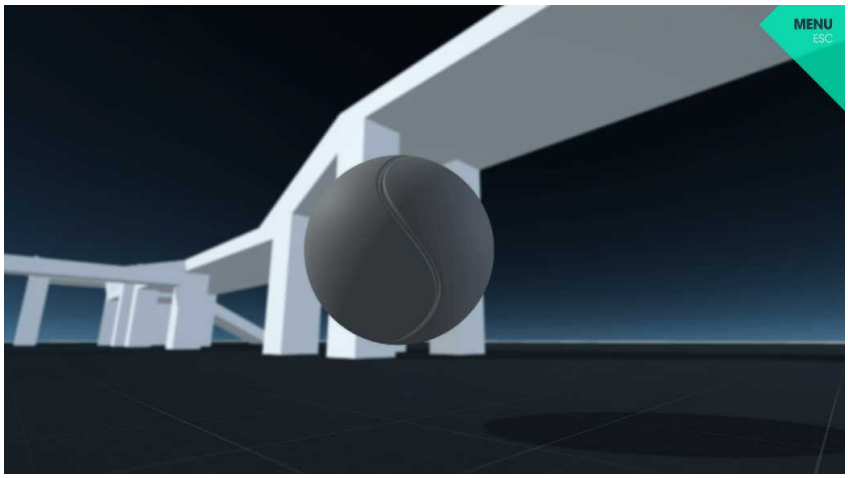
↘ ***CharacterThirdPerson***

Particules

Cet exemple démontre différents effets de particules et de physiques qui peuvent servir dans vos productions.

↘ ***Particles***

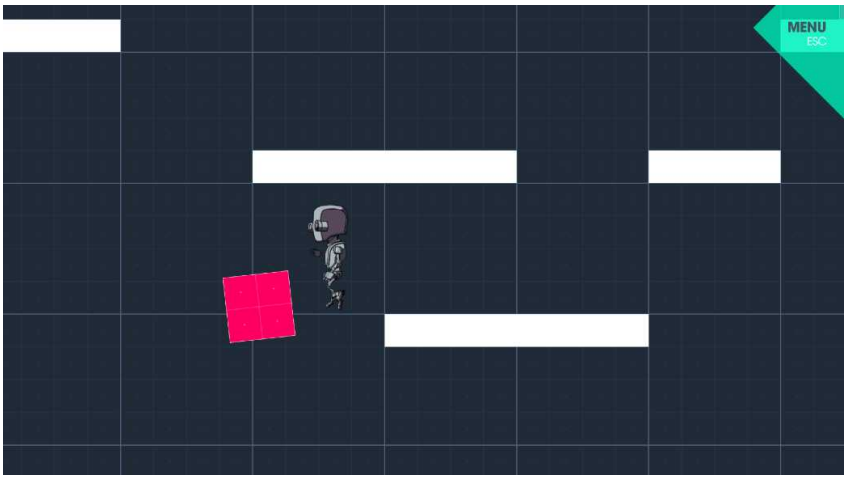
RollerBall



Unity offre un petit jeu de labyrinthe où roule une balle dans un environnement 3D, on voit la qualité de simulation de la physique sur la sphère et la topologie de la scène.

↘ ***RollerBall*** *qui rappelle étrangement le jeu* ***Super Monkey Ball***

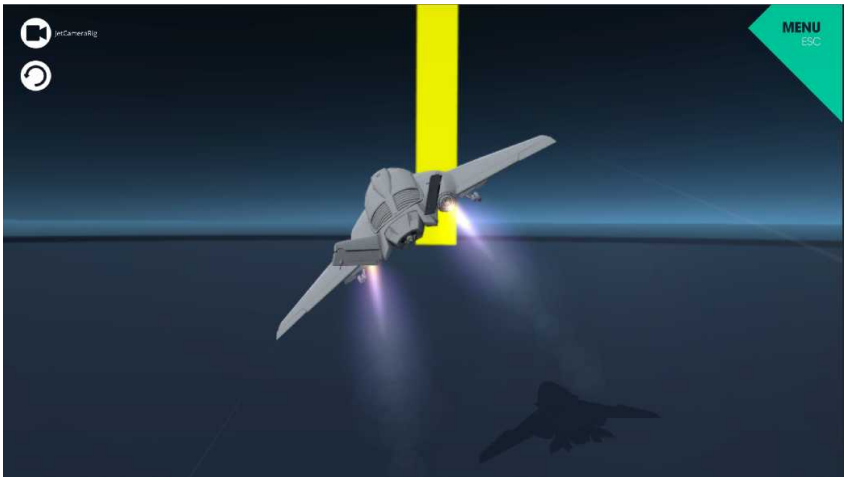
2dCharacter



Depuis la 4e version d'Unity, nous retrouvons un système de jeu 2D pour les amateurs de jeux classiques. Cette démo nous montre les différents types d'interactions 2D que l'on peut utiliser, sans passer par Flash ou un autre logiciel similaire.

↘ ***2dCharacter*** *avec physique et animation de sprites.*

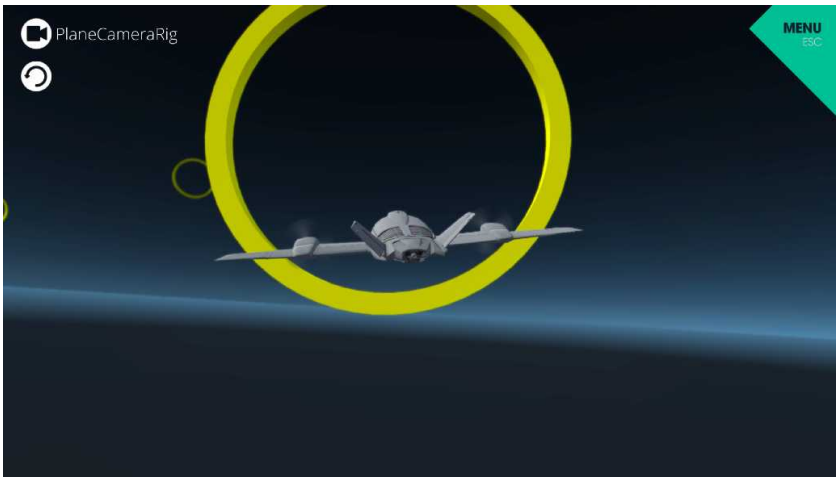
AircraftJet2Axis



Vous avez des voitures comme véhicules, mais également 2 types d'avions, d'abord le jet à réaction, qui se contrôle avec les **flèches** du clavier et qui décélère avec le bouton principal de la souris.

↘ ***AircraftJet2Axis*** *et* ***AircraftJetAI***

AircraftPropeller4Axis



Ensuite, le deuxième type d'avion qui fonctionne avec des hélices, se contrôle avec la souris et accélère avec les flèches avant et arrière du clavier.

↘ ***AircraftPropeller4Axis*** *et* ***AircraftPropellerAI***

Maintenant que vous avez réalisé ce qu'Unity peut offrir, vous allez vous pencher sur les différents outils et menus de l'éditeur.

Apprentissage de la terminologie d'Unity

Comme nous allons référer à ces éléments tout au long du livre, il est préférable de connaître les termes utilisés pour les différentes composantes d'un jeu.

GameObject

Un ***GameObject*** représente tous les objets fondamentaux dans Unity: les personnages ou les éléments des scènes. Le ***GameObject*** ne fait rien en soi, c'est un contenant avec toutes les propriétés d'un objet. Il a besoin de propriétés spaciales pour faire partie d'une scène.

Prefab

Un ***Prefab*** est un contenant qui renferme un ou plusieurs ***GameObjects*** avec leurs propriétés et leurs ***Components***. Les ***Prefabs*** sont utilisés pour créer des instances d'un objet que l'on compte réutiliser dans un jeu. Par exemple, un modèle d'arbre peut servir à créer une forêt.

Assets

Le répertoire ***Assets*** est le répertoire racine du projet. Il renferme tous vos objets (ou ***Assets***).

Component

Les ***Components*** sont essentiellement des scripts avec plusieurs paramètres qui permettent de personnaliser les éléments de votre scène. Les Components les plus fréquemment utilisés sont : ***Transform***, ***Mesh Renderer***, ***Collider***…

Mesh

Un ***Mesh*** représente la géométrie d'un objet. Il est constitué de polygones et peut servir pour définir la topologie ou la surface de collision d'un objet.

Collider

Un ***Collider*** définit la zone de collision d'un objet. Il existe plusieurs types de ***Collider***, (du plus léger au plus lourd) vous y retrouvez : ***Sphere Collider*** (sphère), ***Capsule Collider*** (pilule), ***Box Collider*** (boîte), ***Mesh Collider*** (géométrie complexe)et finalement le ***Wheel Collider*** (roue).

Renderer

Un ***Renderer*** indique au programme si le ***Mesh*** est visible ou non.

Rigidbody

Le ***RigidBody*** définit le comportement de l'objet avec l'engin de physique (nVidia PhysX).

AudioSource

L'AudioSource définit la source d'origine d'un élément sonore dans votre scène.

AudioListener

L'AudioListener définit la position de l'auditeur dans la scène. Seulement un ***AudioListener*** doit être actif dans la scène, sans quoi, le positionnement spacial du son sera incohérent.

Material

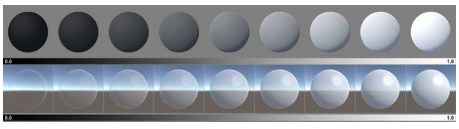
Un ***Material*** (ou matériel) est un contenant qui renferme un ***Shader***, avec des paramètres de couleurs, ***textures*** et effets qui donnent le look des ***GameObjects***.

Shader

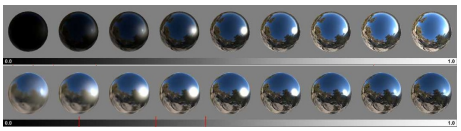
Un ***Shader*** est un programme exécuté par le processeur graphique qui dicte la manière dont les différentes ***textures*** et couleurs vont s'agencer sur un ***Material***. Un ***Material*** doit absolument contenir un seul ***shader.***

Texture

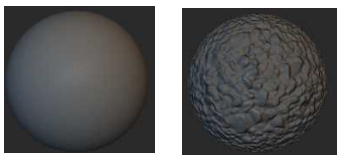
Une ***texture*** est une image 2D qui est appliquée sur un objet. Plusieurs ***textures*** peuvent être combinées pour créer un ***Material*** complexe. Les différentes textures les plus fréquemment utilisées sont :
• **Albedo** (qui remplace **Diffuse**) : Contrôle la couleur de base neutre d'une surface. La transparence fait également partie de l'Albedo. Ses valeurs se mesurent en RVBA (rouge, vert, bleu et transparent).



• **Specular** : Contrôle la réflectivité d'un objet. Ses valeurs sont également en RVBA sauf que la transparence (A) représente le **Smoothness,** c'est-à-dire la diffusion de la lumière.



• **Normal Map** : Contrôle le relief et les bosses que l'on retrouve sur une surface. La texture ***Normal Map*** ne modifie pas la topologie de l'objet mais seulement la manière dont la lumière doit réagir sur la surface d'un objet. En d'autres mots, ce n'est qu'une illusion de relief.



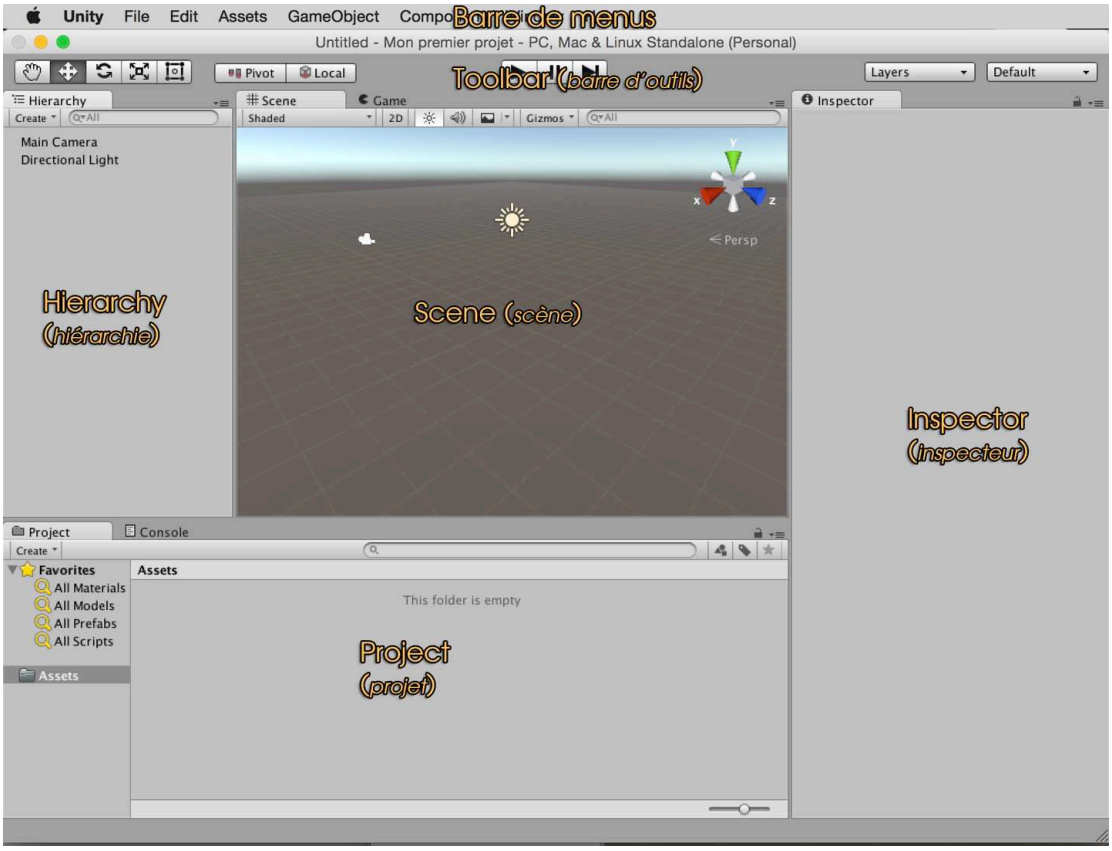
• **Emission** : Contrôle la couleur et l'intensité de la lumière émise par la surface.

D'autres concepts seront introduits tout au long du document.

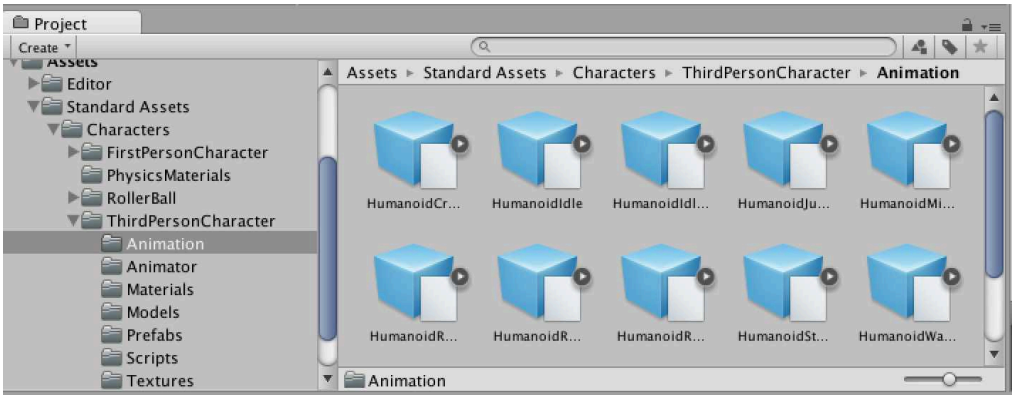
Apprentissage de l'interface

Prenez un peu de temps pour vous familiariser avec l'interface de l'éditeur. Par défaut, la fenêtre principale de l'éditeur est constituée de plusieurs panneaux assemblés. Dans Unity, ces panneaux sont officiellement appelés des ***Views*** (vues).

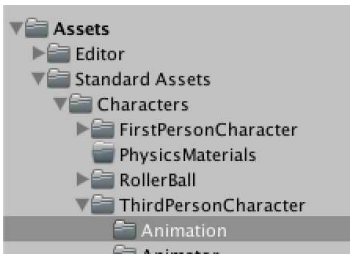
Unity comprend plusieurs de ces panneaux qui ont des rôles spécifiques décrits dans cette section.



Panneau (Project browser)



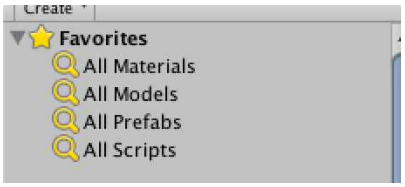
Par défaut, ce panneau est séparé en deux colonnes, À gauche, vous avez une vue hiérarchisée de l'arborescence des répertoires de votre projet. Quand vous sélectionnez un de ces répertoires, son contenu sera révélé dans la colonne de droite.



Les flèches dans la colonne de gauche permettent d'afficher ou de cacher la structure d'un répertoire. Si vous appuyez sur une flèche en tenant le bouton du clavier enfoncé (**Alt**) sur Windows ou (⌥) sur Mac, vous pouvez afficher ou cacher tous les sous-répertoires récursivement.

Le contenu individuel d'un répertoire est affiché dans la colonne de droite et l'icône illustre leur type d'objet (**Script, matériel, sous-répertoire, etc**.). Ces icônes peuvent être redimensionnées en utilisant la barre de défilement qui se trouve dans le coin, en bas à droite du panneau.

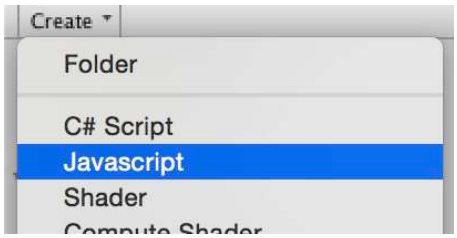
À la gauche de cette barre de défilement, vous y retrouvez le nom de l'élément actuellement sélectionné.



Dans la colonne de gauche, vous y retrouverez également une section *Favorites* où vous pouvez conserver les éléments les plus fréquemment utilisés pour un accès rapide. Pour y ajouter un élément, vous n'avez qu'à le glisser à partir de *Project,* dans la section *Favorites*.

Au-dessus de la colonne de droite, vous verrez s'afficher le chemin complet du répertoire. Vous pouvez naviguer dans les sous-répertoires en cliquant sur un des liens.

Dans le haut du panneau, on retrouve la barre d'outils du *Project browser*.



Dans cette barre d'outils, vous y retrouvez un bouton « Create » pour accéder rapidement aux outils de créations d'assets. Ce n'est qu'une des méthodes pour créer des: (**scripts, matériaux, masques, etc.**)

Notez-bien!

Vous pouvez également faire un clic secondaire dans le panneau afin de créer un nouvel élément.

Complètement à droite de ce panneau, on retrouve le menu de la fenêtre qui permet de changer l'affichage du panneau en une ou deux colonnes. Le mode « une colonne » affiche le panneau sous forme d'une arborescence simple où les éléments sont listés sous la forme d'un champ texte, sans aperçu du contenu.

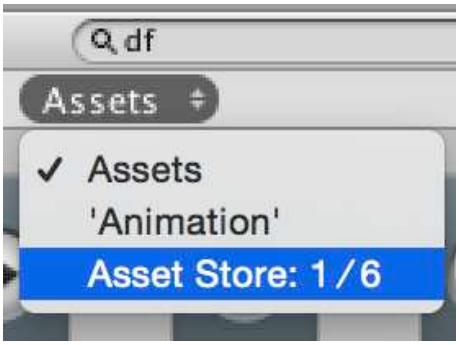
Conseil!

Il est préférable d'utiliser le mode à une seule colonne, le panneau requiert alors moins d'espace dans l'interface.

À gauche de ce bouton, il y a un cadenas qui permet d'empêcher un changement.

Rechercher un élément dans le projet

Vous retrouvez également un outil de recherche particulièrement utile quand un projet contient un bon nombre d'assets et lorsque vous n'êtes pas familiers avec celui-ci. La recherche de base va filtrer les éléments selon le texte inscrit.

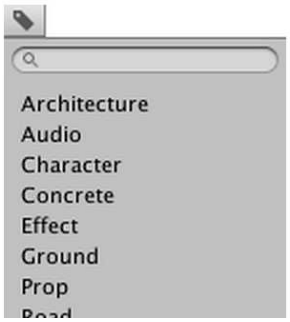


En-dessous du champ de recherches, vous avez la possibilité de changer l'emplacement de cette recherche si vous appuyez sur « Assets ».

Cette section offre trois choix de recherches :
- la racine du répertoire *Assets*,
- le répertoire sélectionné
- le magasin en ligne *Asset Store*

(Incluant le nombre d'éléments gratuits et payants séparés par « / »).

À la droite du champ de recherches, vous retrouvez trois boutons de filtres. Le premier bouton permet de filtrer les résultats de la recherche selon le type d'éléments (***Animation***, ***AudioClip***, ***Font***, etc.)



Le second bouton utilise les étiquettes (***Label***). Les étiquettes peuvent être assignées aux éléments dans le panneau *Inspector*. Comme un projet peut contenir plusieurs étiquettes, un petit champ de recherches est intégré à ce filtre.

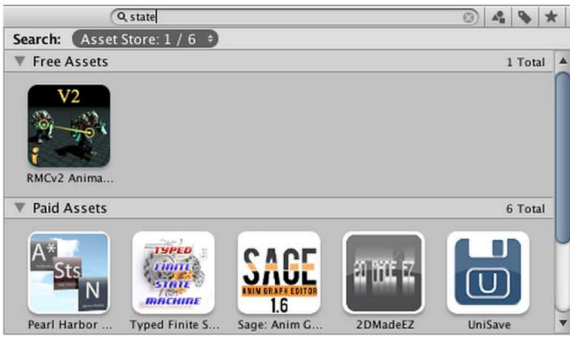
Lorsque vous ajoutez un filtre, un nouveau terme sera ajouté au champ de recherche. Un terme débutant par « t : » filtrera les éléments selon leur type, « l : » filtre par étiquette (***Label***). Vous verrez ces termes directement dans le champ afin que vous puissiez rapidement en ajouter ou retirer, sans devoir passer par un menu. L'utilisation de plusieurs filtres permettra de réduire le nombre de résultats, car ces termes sont additifs, c'est-à-dire que toutes les conditions doivent être remplies afin d'afficher un élément dans la liste des résultats.

Finalement, le dernier bouton en forme d'étoile permet de sauvegarder les résultats d'une recherche dans *Favorites.*

Recherche dans le magasin *Asset Store*

Non seulement vous pouvez faire une recherche détaillée dans votre projet, mais vous pouvez également faire une recherche directement sur le magasin en ligne.

Lorsque vous inscrivez quelque chose dans le champ de recherche, par défaut, celle-ci se fait à la racine du répertoire *Assets*. Si vous changez cette valeur pour : **Asset Store X/Y,** vous pouvez voir les



sources disponibles en ligne, séparées selon le prix (*gratuit ou payant*). Les éléments seront également rangés selon leur popularité.

Si vous sélectionnez un élément de cette liste, vous verrez ses détails dans *l'Inspector* avec l'option d'acheter ou de télécharger ces composantes. Vous avez également la possibilité de voir un aperçu de l'élément en question, avant de l'acheter.

Les raccourcis clavier

La liste des raccourcis suivants est fonctionnelle quand le focus est placé sur le panneau *Project*. (Pour placer le focus, il faut positionner le curseur de la souris au-dessus du panneau ou de la fenêtre).

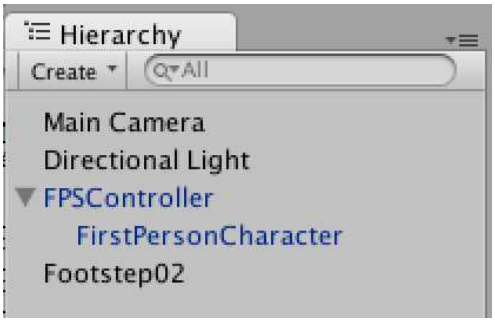
Notez-bien!

Certains raccourcis sont seulement disponibles en mode deux-colonnes (Two-columns).

Raccourcis : Panneau Project	
F	Montre l'élément sélectionné dans son répertoire (utile lors d'une recherche)
Tab ⇥	Change le focus entre les deux colonnes (en mode two-columns seulement)
CTRL+ F (Win) / ⌘ + F (Mac)	Mets le focus sur le champ de recherche
CTRL+ A (Win) / ⌘ + A (Mac)	Sélectionne tous les éléments visibles dans la liste
CTRL+ D (Win) / ⌘ + D (Mac)	Duplique les éléments sélectionnés
Supprim.	Supprime l'élément sélectionné avec confirmation
⇧ + Supprim.	Supprime l'élément sélectionné sans confirmation
⌘ + ⌫ (Mac)	Supprime l'élément sélectionné sans confirmation (OS X)
F2 (Win)	Renomme l'élément sélectionné (Windows)
↩ (Mac)	Renomme l'élément sélectionné (OS X)
⌘ + ↓ (Mac)	Ouvre l'élément sélectionné dans son programme par défaut (OS X)

Raccourcis : Panneau Project	
Entrer (Win)	Ouvre l'élément sélectionné dans son programme par défaut (Windows)
⌘ + ↑ (Mac)	Sélectionne le répertoire parent (OS X en mode two-columns seulement)
Espacement arrière (Win)	Sélectionne le répertoire parent (Windows en mode two-columns seulement)
→	Affiche l'arborescence du répertoire sélectionné ou sélectionne le premier élément d'une liste, si le contenu est déjà affiché.
←	Ferme l'arborescence du répertoire sélectionné ou sélectionne son parent, si le contenu est déjà caché.
ALT + → (Win)/⌥+ → (Mac)	Affiche le contenu d'un élément sélectionné
ALT + ← (Win)/⌥+ ← (Mac)	Cache le contenu d'un élément sélectionné

Panneau (Hierarchy)

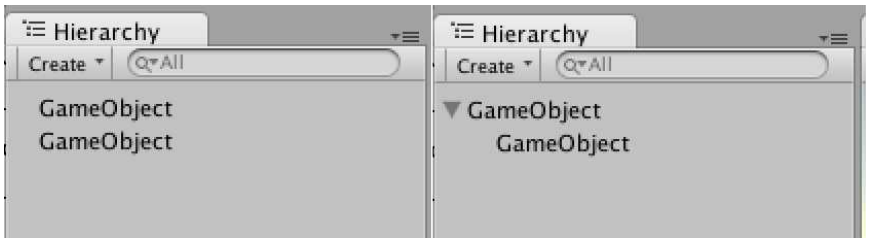


Le panneau *Hierarchy* contient tout le contenu de la scène active. Certains de ces éléments sont des objets créés directement dans la scène. D'autres, sont des copies de ***Prefabs***, qui sont des ***GameObjects*** réutilisables qui coexistent dans la scène et le projet simultanément. Vous pouvez sélectionner des objets directement dans le panneau *Hierarchy,* les glisser sur d'autres objets afin de changer leur parenté. Quand des éléments sont ajoutés ou retirés de la scène, ceux-ci se reflètent également dans *Hierarchy*.

La parenté des objets

Unity® utilise le concept de parenté afin de créer une interdépendance entre plusieurs objets. Pour rendre un ***GameObject*** enfant d'un autre, glissez celui-ci sur le parent désiré dans *Hierarchy*.

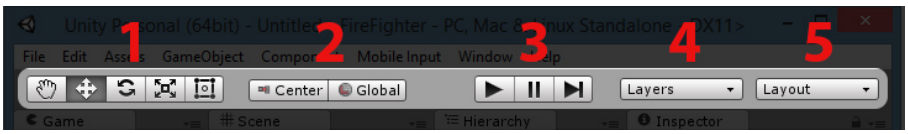
L'enfant héritera du mouvement, de la rotation et des proportions de son parent, ses coordonnées seront relatives au parent. Lorsque les deux objets ont un lien de parenté, on peut dévoiler les enfants en cliquant sur la flèche à gauche du nom du parent dans *Hierarchy*. Si vous enfoncez la touche (**Alt**) ou (⌥) sur Mac, au clavier, et que vous cliquez sur un parent, tous ses enfants et ses descendants seront affichés ou cachés.



L'image précédente illustre à gauche deux objets indépendants et à droite les ***GameObjects*** ayant un lien de parenté.

La barre d'outils (Toolbar)

La barre d'outils contrôle cinq groupes de boutons qui servent à manipuler différentes parties de l'éditeur.



Les outils de transformations servent à interagir avec les objets qui apparaissent dans la fenêtre de la scène (*Scene View*).

Les modificateurs du (*Transform Gizmo*) affectent le comportement des pivots dans la scène.

Les boutons : « Play »/« Pause »/« Step » sont utilisés pour démarrer et manipuler le déroulement du jeu, dans la fenêtre *Game View*.

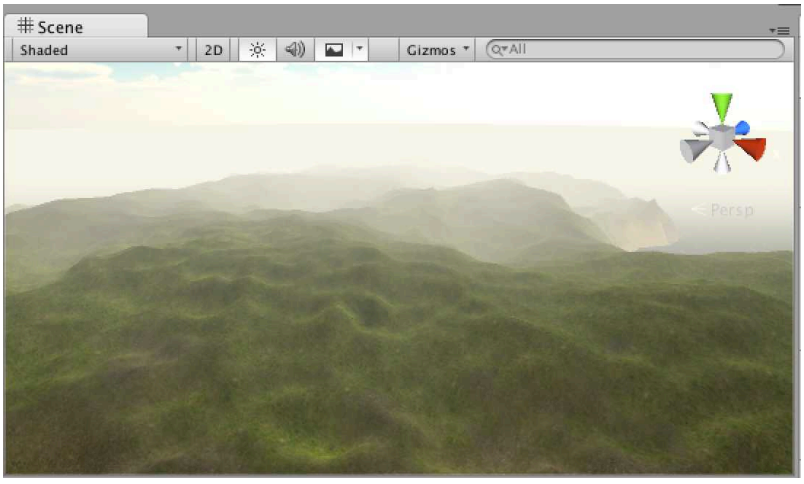
La liste défilante ***Layers*** contrôle l'affichage des différents groupes objets dans la scène.

La liste défilante ***Layout*** contrôle l'arrangement de tous les panneaux dans l'éditeur.

Plus tard, vous reviendrez sur ces outils, pour l'instant, j'attire votre attention vers la fenêtre *Scene View*.

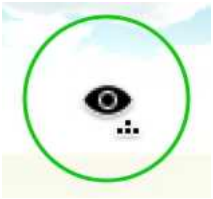
La fenêtre (Scene View)

La fenêtre de la scène est votre chantier de construction interactif. Vous utilisez *Scene View* pour sélectionner ou pour positionner les différents éléments qui constituent votre environnement (**les avatars, les caméras, les lumières, et autres *GameObjects***).



Naviguer dans la scène

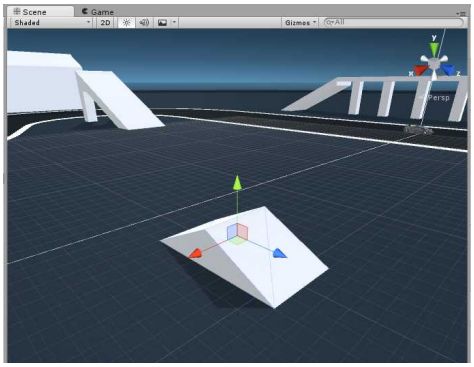
Voici un bref aperçu des contrôles essentiels pour travailler dans la fenêtre de la scène :



• Maintenez enfoncez le **bouton secondaire** de votre souris afin de survoler votre scène. Lorsque vous jumelez ce bouton avec les touches du clavier : **W, A, S, D** (plus **Q** et **E** pour monter et descendre), vous pouvez naviguer dans votre scène, comme un jeu à la première personne (*ex. : FPS*).

Notez-bien!

*Pour ceux et celles qui utilisent un clavier **AZERTY**, au lieu de la disposition anglo-saxon **QWERTY**, veuillez consulter la rubrique sur la personnalisation des touches du clavier afin d'ajuster celles-ci correctement.*



• Sélectionnez un objet et appuyez sur la touche **F** en survolant la fenêtre *Scene View* pour centrer le **focus** sur le pivot de cet objet.

• Utilisez les **flèches directionnelles** au clavier pour vous déplacer en longueur et en largeur. (*Ces touches ne fonctionnent pas en mode survol!)*

• Maintenez **la touche** (**Alt**) ou (⌥) enfoncée sur Mac, tout en **cliquant avec le bouton principal** et en **déplaçant votre souris** pour pivoter en orbite dans votre scène.

Notez-bien!

*La différence entre ce mode et le mode survol, c'est qu'en survol, on peut se déplacer avec les touches **WASD**, mais pas dans ce cas-ci.*

• Maintenez **la touche** (**Alt**) ou (⌥) enfoncée sur Mac tout en **cliquant avec le bouton du milieu** et en **déplaçant votre souris,** pour modifier votre point de vue en translation dans votre scène.

• Maintenez **la touche** (**Alt**) ou (⌥) enfoncée sur Mac tout en **cliquant avec le bouton secondaire** et en **déplaçant votre souris,** pour zoomer vers l'avant ou l'arrière. (*C'est le même comportement en utilisant la roulette de votre souris*)

Vous pouvez également utiliser l'outil de la main (**raccourci** : **Q**) pour accomplir les mêmes résultats, ce qui est particulièrement utile si vous utilisez une souris avec un seul bouton.

Avec l'outil de la main activée :

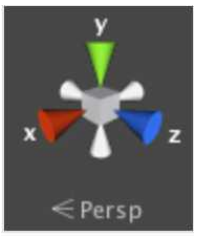
• **Cliquez** et **déplacez votre souris** pour déplacer votre point de vue en translation.

• Maintenez la touche (**Alt**) ou (⌥) enfoncée sur Mac, **cliquez** et **déplacez votre souris** pour bouger la caméra en orbite sur son pivot actuel.

• Maintenez la touche (**Alt**) ou (⌥) enfoncée sur Mac, **cliquez** avec le **bouton secondaire** et **déplacez votre souris** pour zoomer vers l'avant ou l'arrière.

Notez-bien!

*Sur Mac, vous pouvez également utiliser les touches (**Ctrl +** ⌥), cliquez et déplacez votre souris pour zoomer avec une souris à un bouton.*



L'utilisation du *Gizmo*

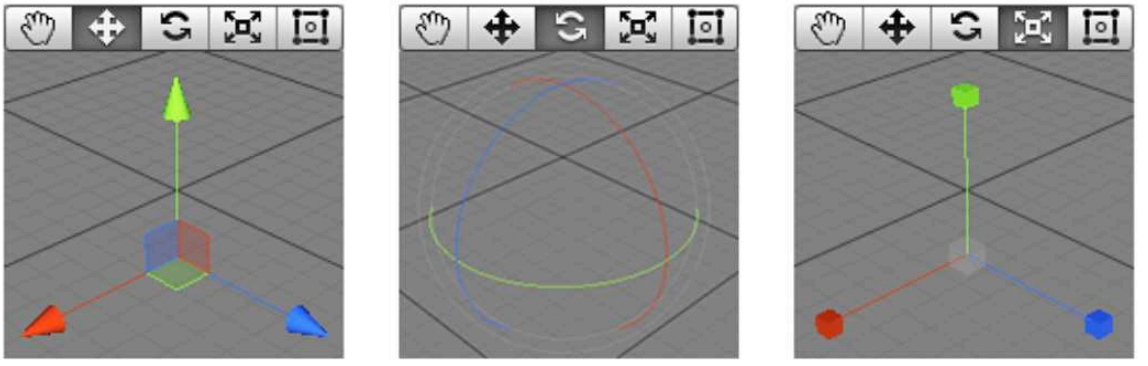
Dans le coin supérieur droit de la fenêtre *Scene View,* vous y retrouvez le ***Gizmo*** de la scène. Cet élément montre l'orientation de votre point de vue dans la scène. En cliquant sur les différents axes du ***Gizmo***, vous pouvez rapidement changer votre point de vue.
Chaque pointe de couleurs représente un axe géométrique. Vous pouvez cliquer sur un axe afin de positionner votre caméra dans un angle droit. Vous pouvez également appuyer sur le texte en-dessous du ***Gizmo*** pour changer la perspective de la caméra en mode isométrique (sans profondeur de champ) et vice-versa.

Notez-bien!

*Le **Gizmo** est seulement disponible pour les jeux en 3D. Si votre jeu est configuré en 2D, le **Gizmo** devient alors inutile.*

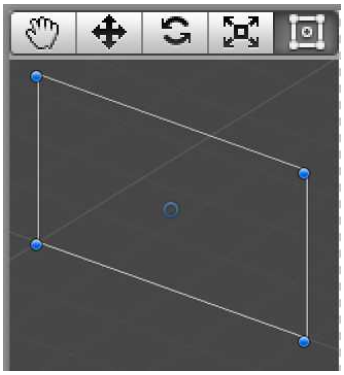
Transformation des *GameObjects*

Quand vous assemblez vos scènes, vous utilisez plusieurs objets de natures différentes. Pour accomplir cette tâche, vous utilisez les outils de transformations (***Transform Tools***) dans la barre d'outils pour faire des translations, rotations et pour redimensionner les ***GameObjects***. Chaque outil a son propre ***Gizmo*** qui apparaît autour de votre sélection, dans la scène (*Scene View*). Vous pouvez manipuler les axes des ***Gizmos*** pour modifier les propriétés des objets sélectionnés.



↘ ***Gizmos,*** *les trois outils de transformations : translation* ***(W)****, rotation* ***(E),*** *redimensionnement* ***(R).***

L'outil *Rect tool)*



L'outil ***Rect tool*** est une nouveauté d'Unity 4.6. On retrouve cet outil au bout de la liste d'outils de transformations. Il permet d'éditer les ***sprites*** et objets 2D comme les menus et les interfaces utilisateurs.

Pendant le redimensionnement d'un objet 2D (***sprite***), vous pouvez tenir la touche (⇧) enfoncée pour maintenir les proportions de l'objet. Vous pouvez également utiliser l'outil ***Rect tool*** sur les objets 3D, les poignées sont affichées selon votre point de vue dans la scène.

↘ *L'outil Rect* ***(T)***

La barre de contrôle de la fenêtre (*Scene View*)

Ces contrôles vous permettent de changer plusieurs options pour l'affichage de la fenêtre *Scene View*. Vous pouvez activer/désactiver le son, l'éclairage et passer en mode 2D. Vous pouvez également contrôler l'affichage des objets, des effets spéciaux (***image effects***) et des ***Gizmos***.

La fenêtre (Game View)

La fenêtre *Game View* vous démontre ce que les joueurs verront en finalité, dans la version publiée. Vous devez avoir au moins une caméra dans la scène.

Les boutons de test

- ***Play*** : Démarre le jeu dans l'éditeur.
- ***Pause*** : Interrompt le jeu sans l'arrêter.
- ***Next Frame*** : Lance le jeu pour une image avec l'option d'interrompre le jeu à nouveau.

Utilisez ces boutons dans la barre d'outils pour tester votre jeu durant son développement.

Notez-bien!

*Dans **Hierarchy**, pendant le déroulement du jeu en mode (**Play**), tous les changements ne seront pas conservés une fois la session de jeu terminée. L'éditeur change la couleur afin de vous rappeler quel mode est actif.*

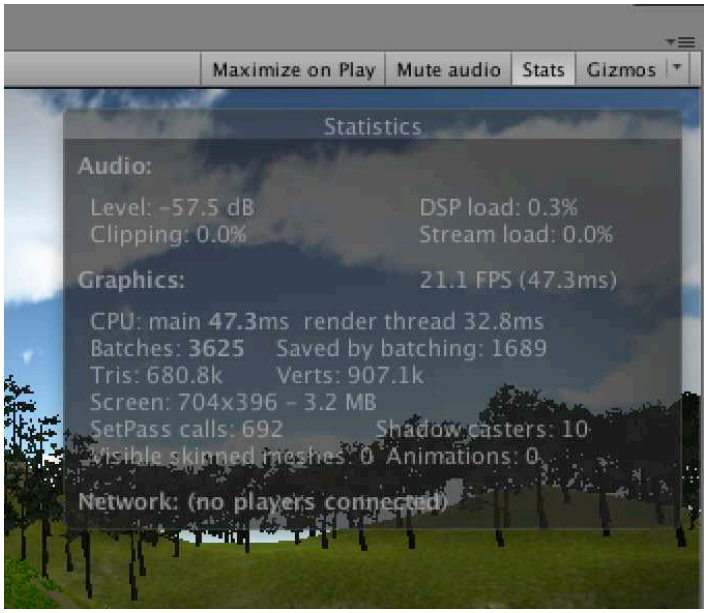
La barre de contrôle de la fenêtre (*Game View*)

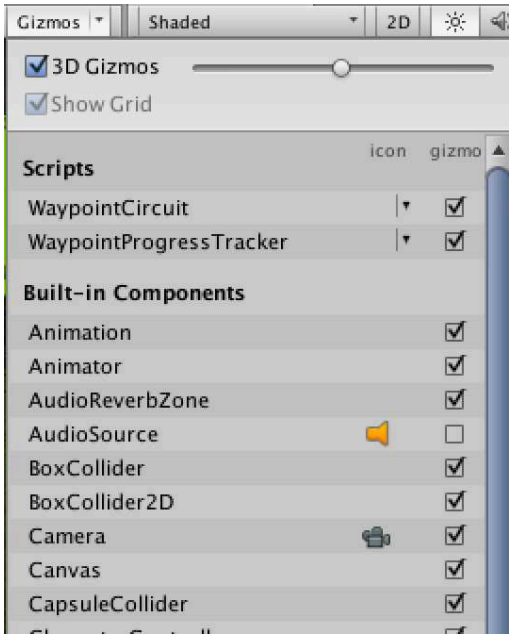
Le premier menu, déroulant de la barre de contrôle, vérifie les proportions de l'écran. C'est à cet endroit que vous pouvez spécifier un ratio pour la fenêtre *Game View*. Ceci est utile lorsqu'on veut tester l'apparence d'un jeu sur différents types d'écran.

Plus loin, à droite, vous avez l'interrupteur « Maximise on Play » pour maximiser la taille de la fenêtre *Game View* quand on est en mode (**Play**). Ceci cache toutes formes de distractions dans l'éditeur lorsque l'on teste le jeu.

Ensuite, on retrouve l'interrupteur « Mute audio » qui coupe le son durant le déroulement du jeu.

Le troisième interrupteur, vers la droite, « Stats » affiche les statistiques des performances de votre jeu ainsi que d'autres informations sur le rendu de votre scène. Cet outil est fort utile pour surveiller les pertes de performances pendant le jeu.





Le dernier bouton est l'interrupteur des ***Gizmos***. Quand celui-ci est actif, tous les ***gizmos*** qui apparaissent dans la fenêtre *Scene View* seront également affichés dans *Game View*. Ce bouton possède également un menu qui inclut les différents ***Components*** qui sont utilisés dans le jeu.

• Dans ce menu, vous retrouvez également une colonne nommée (**icon**) qui permet de personnaliser un ***Component*** avec un icône.

En appuyant sur l'une de ces flèches, un nouveau menu apparaîtra, vous permettant de choisir l'icône désiré.

• Dans le menu précédent, le paramètre (***gizmo***) permet de désactiver l'affichage du ***gizmo*** d'un component spécifique.

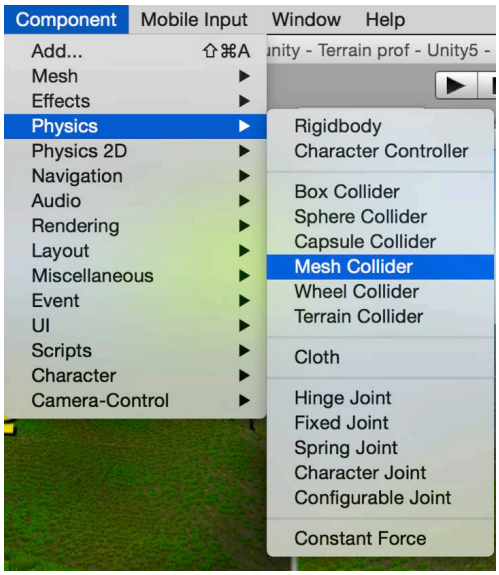
Le paramètre ***3D Gizmos*** dans le haut du menu fait référence aux icônes ***gizmos***. Lorsqu'elles sont activées, les icônes vont être dimensionnées selon leur distance de la caméra. Autrement, tous les icônes pour les objets auront la même taille, peu importe leur distance. La barre de défilement suivant *3D Gizmos* permet d'ajuster la taille des *Gizmos* dans ***Game View,*** ce paramètre peut s'avérer utile lorsqu'il y a trop d'icônes visibles à l'écran.

Le panneau (Inspector)

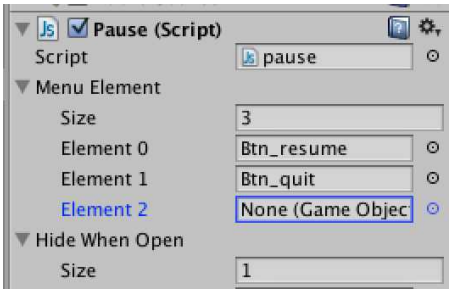
Les jeux sont constitués de plusieurs ***GameObjects,*** qui eux-mêmes, contiennent des ***Meshes***, des ***scripts***, et parfois d'autres éléments visuels optionnels comme un ***AudioSource*** ou une **lumière**. *L'inspector* affiche des informations détaillées sur le ***GameObject*** sélectionné, incluant tous les ***Components*** et leurs propriétés. C'est ici que vous modifiez les fonctionnalités des ***GameObjects*** dans votre *Scene View*. La relation entre les ***GameObjects*** et ses ***Components*** est importante à connaître.

Les (*Components*)

Vous pouvez ajouter un nouveau ***Component*** à partir de la barre de menus, en appuyant sur : (***Component***)**.** De là, vous avez plusieurs sous-menus permettant de choisir les éléments nécessaires.



Vous pouvez également vous servir du bouton « Add Component » qui se trouve dans le bas de *l'Inspector*.



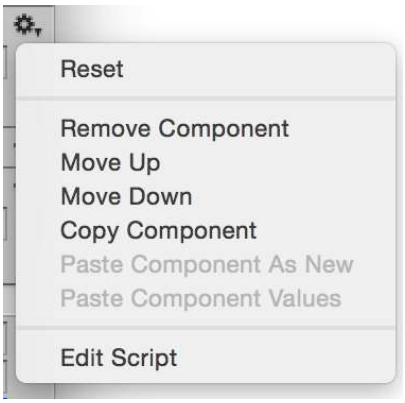
Toutes les propriétés qui sont affichées dans *l'Inspector* peuvent être directement modifiées sans devoir alterner le script.

Afin d'expérimenter, lors de l'exécution du programme, vous pouvez utiliser *l'Inspector* pour changer des ***variables*** et ainsi trouver les meilleures valeurs à appliquer à celles-ci, après son exécution. Une fois le jeu arrêté, tous les changements dans la scène seront réinitialisés.

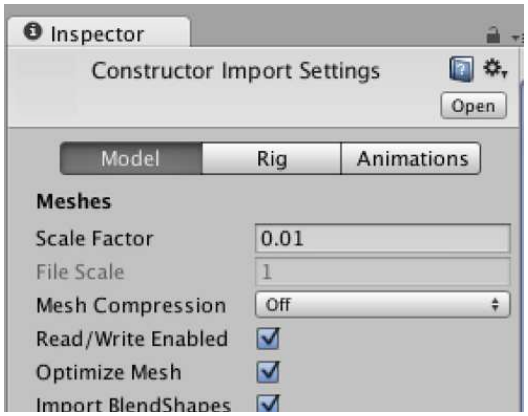
Dans un script, vous pouvez définir des ***variables*** publiques d'un type d'objet (par exemple : ***GameObject***, ***RigidBody*** ou ***Transform***) et ensuite glisser des objets dans *l'Inspector* pour assigner une valeur à la variable.

Si vous désirez plus de détails sur un ***Component***, vous pouvez obtenir la **documentation** de celui-ci en appuyant sur le **livre avec un point d'interrogation** qui se trouve à la droite du nom de ce ***Component***.

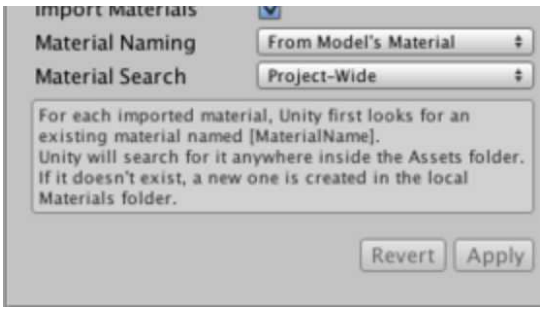
Si vous appuyez sur l'icône de **l'engrenage** (ou si vous faites un clic secondaire sur un ***Component***), un menu contextuel spécifique apparaîtra.



Ce menu permet de retirer, réinitialiser, déplacer, copier, coller les valeurs et éditer le ***Component***.



L'Inspector vous montre également les paramètres importés d'un élément quand on le sélectionne dans le panneau *Project*.



Notez-bien!

Quand vous modifiez les paramètres d'un élément importé, appuyez sur le bouton « Apply » pour confirmer vos changements.

C'est également dans *l'Inspector* que vous assignez un marqueur (***Tag)*** et un calque (***Layer***) pour vos éléments.

Les (*Prefabs*)

Trois boutons additionnels sont également présents quand un ***prefab*** est sélectionné. Ces boutons permettent de modifier toutes les instances du ***prefab*** simultanément.

- Le bouton « Select » sélectionne *l'élément* original dans le panneau *Project*.
- Le bouton « Revert » annule les changements apportés au prefab et réinitialise les valeurs originales du prefab.
- Le bouton « Apply » permet de modifier toutes les instances du prefab à partir de l'instance sélectionnée.

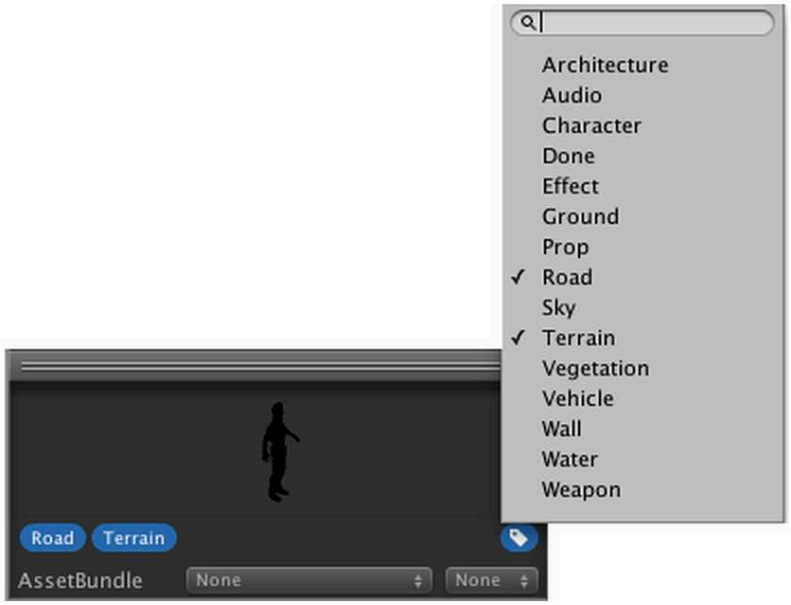
Les icônes (*Icons*)

Une icône personnalisée peut être définie pour identifier l'objet dans la scène. Si vous cliquez sur l'icône dans *l'Inspector*, un menu de sélection s'affiche.

Les étiquettes (*Labels*)

Unity vous permet d'utiliser des mots-clés sous forme d'étiquettes (***Labels***) pour rendre ces éléments plus faciles à repérer. Dans le bas de *l'Inspector,* vous avez un aperçu de votre objet. Dans le coin en bas à droite, vous retrouvez le bouton pour ajouter une étiquette.

En appuyant sur ce bouton, un menu avec les différents mots-clés s'affichera.



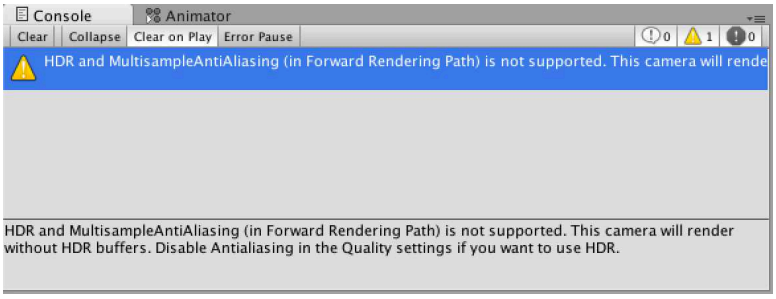
Vous pouvez choisir un ou plusieurs mots-clés qui seront marqués par un crochet. (Ils se retrouvent également inscrits dans le panneau d'aperçu). Si vous sélectionnez ces étiquettes à nouveau, le mot-clé sera retiré de l'objet.

Vous pouvez également rechercher ou ajouter un nouveau mot-clé en utilisant le champ de recherche qui se trouve dans le haut du panneau. Si un mot n'existe pas, il sera ajouté à la liste lorsque vous appuyez sur la touche (**Entrée/Retour)** de votre clavier.

Une fois appliqués, ces mots-clés vous permettront de retrouver ces objets plus rapidement, lors d'une recherche.

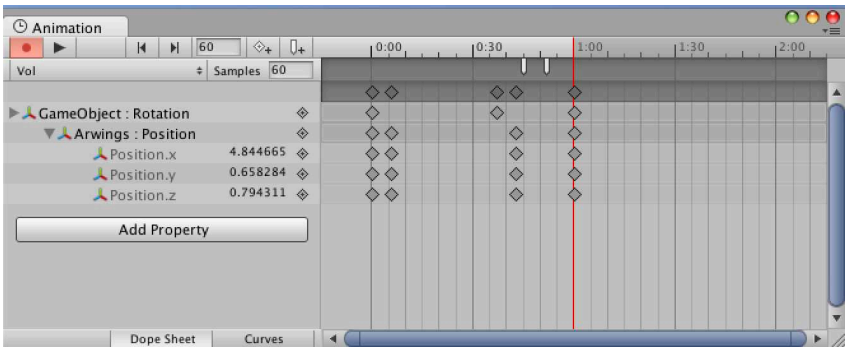
Les fenêtres (Views) additionnelles

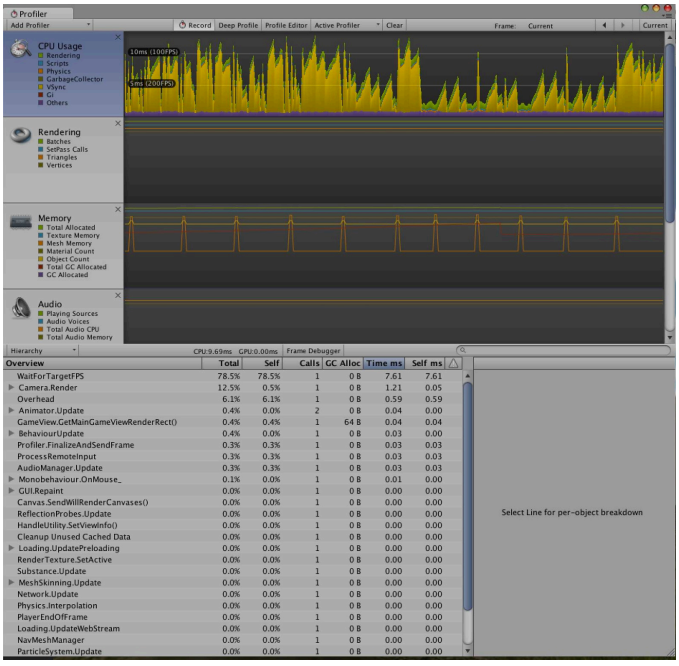
Il existe d'autres fenêtres et outils de base dans l'interface d'Unity. Ceux-ci comprennent:



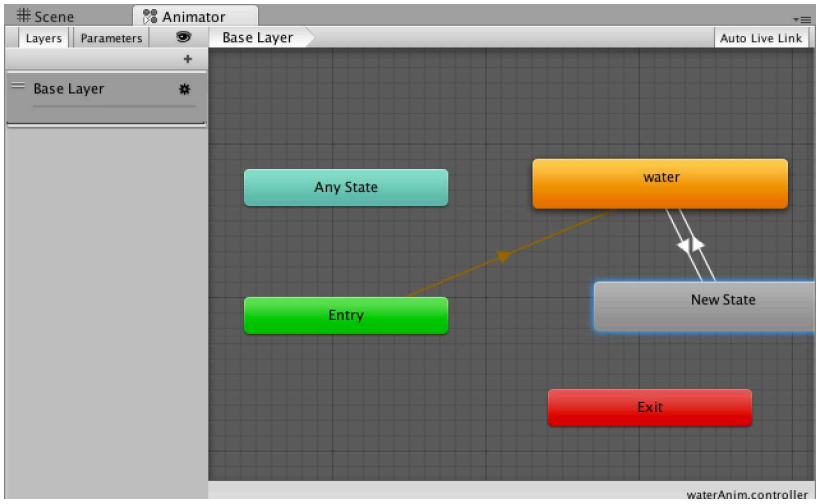
↘ La *Console* qui affiche les messages d'alertes ou d'erreurs.

La fenêtre *Animation View* pour animer vos objets dans la scène. ↙

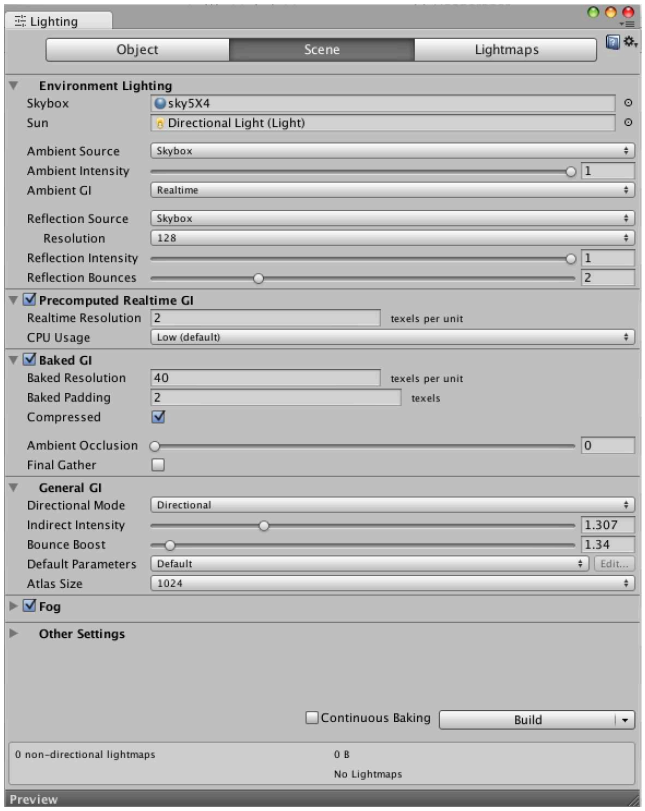




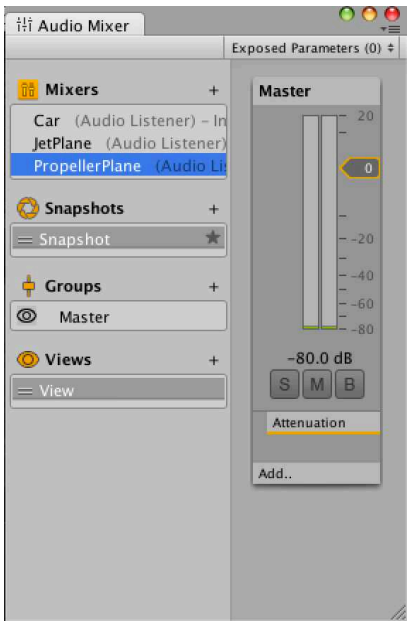
↘ Le *Profiler* sert à analyser les performances du jeu durant son exécution, afin d'identifier les problèmes.



La fenêtre *Animator* à ne pas confondre avec *Animation,* sert à assembler et mixer les différents clips d'animations sous forme d'une arborescence pour l'intégration dans le jeu.



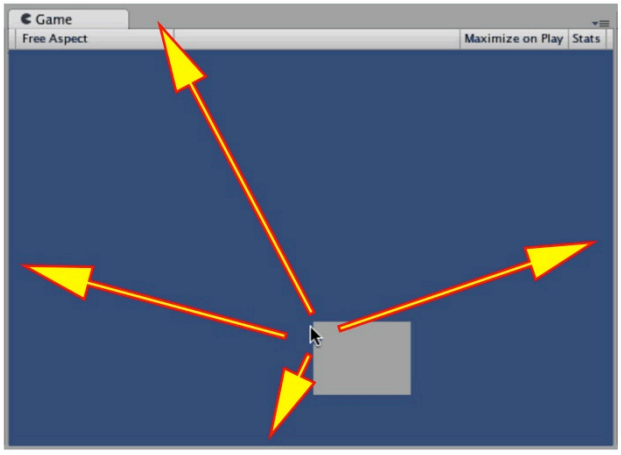
La fenêtre *Lighting* gère l'éclairage de l'environnement, le ***skybox,*** le soleil, l'illumination globale, le brouillard, l'occlusion ambiante et plus.



La console *Audio Mixer* est nécessaire pour changer le volume et les effets des différents éléments sonores de votre jeu.

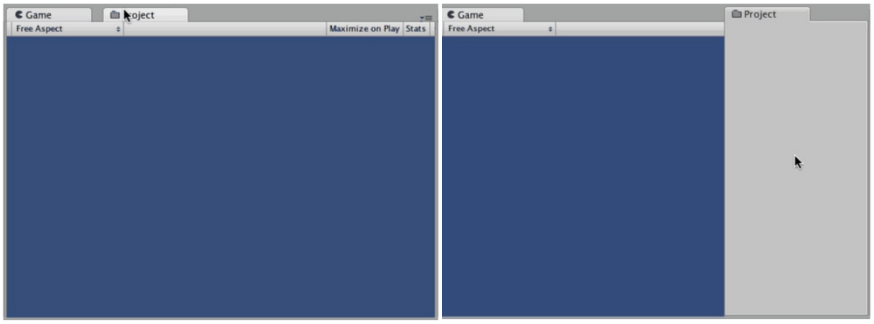
Pour plus de détails sur les différents outils d'Unity, veuillez consulter le site Web de la documentation d'Unity.

Personnalisation de votre espace de travail

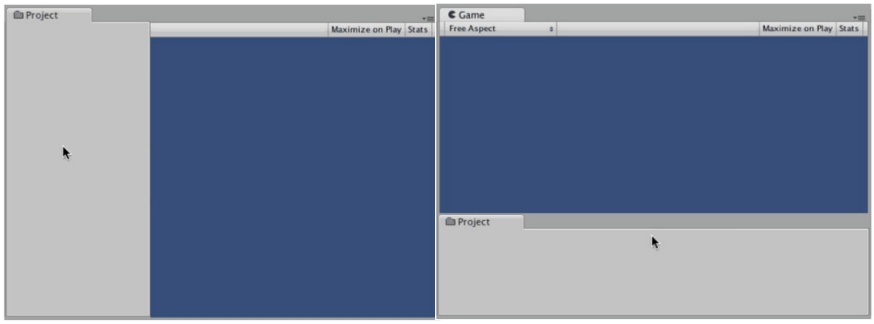


Vous pouvez changer la configuration de votre espace de travail afin de l'ajuster selon vos goûts et besoins.

Vous n'avez qu'à glisser et ancrer les différents panneaux à l'une des extrémités de l'interface. Si vous lâchez un panneau dans une barre d'onglets, celui-ci sera placé derrière un panneau existant. Vous pouvez également glisser les panneaux et les onglets à côté d'un autre pour ancrer sa position à l'écran.



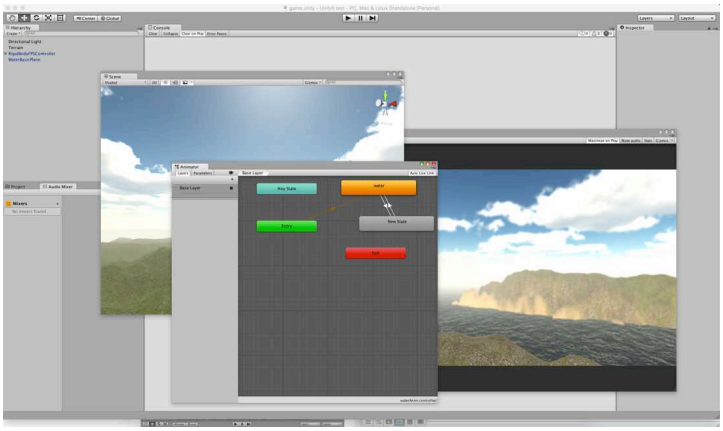
↘ Ancrage dans l'onglet et à droite de la fenêtre

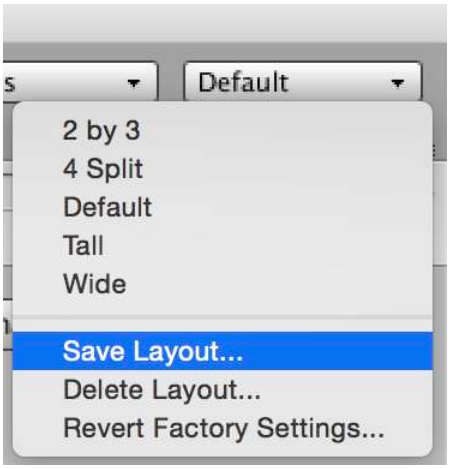


↘ Ancrage à gauche et dans le bas de la fenêtre

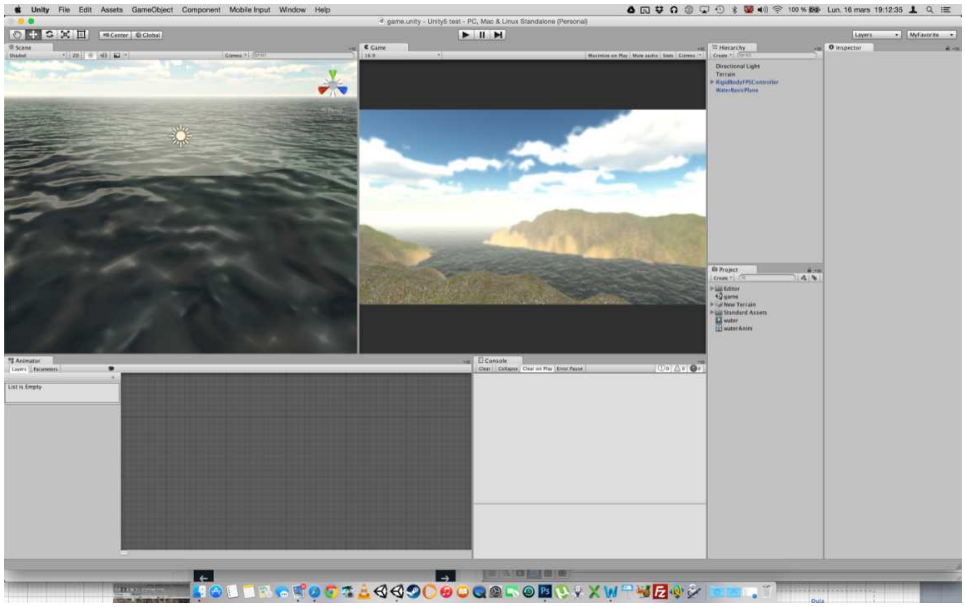
Les panneaux peuvent également être détachés de l'interface afin de créer des fenêtres flottantes qui peuvent être disposées individuellement sur un ou plusieurs moniteurs.

Fenêtres flottantes ↙

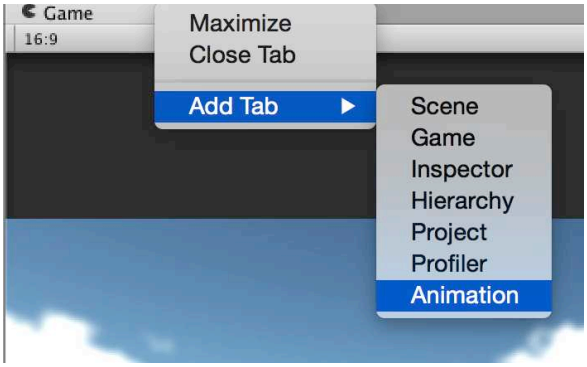




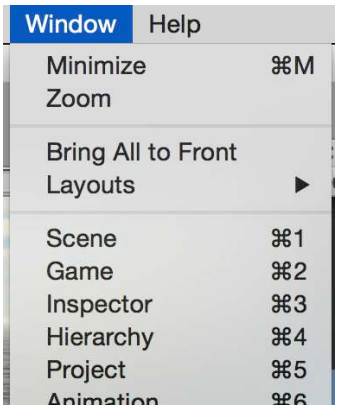
Lorsque vous êtes satisfaits de la disposition de votre interface, vous pouvez la sauver afin de pouvoir y revenir plus tard. Dans la barre d'outils, appuyez sur le bouton « Layout » qui se trouve dans le coin supérieur droit de l'éditeur. En appuyant sur ce bouton, une liste déroulante apparaîtra. Vous pouvez ajouter une nouvelle disposition en appuyant sur « Save Layout… ». Vous pourrez y revenir en tout temps, à partir de cette liste.



Interface complètement personnalisée.



Vous pouvez ajouter un panneau à tout moment, simplement avec un clic secondaire sur un onglet ou à partir de la barre de menus : **Window.**



Le processus de création des textures

Lorsqu'on travaille avec des textures dans un jeu vidéo, il est important de connaître quelques règles. Une mauvaise utilisation des textures peut mener à des jeux beaucoup trop lourds (en mémoire *RAM* et en espace de stockage), sans compter les délais de compression et de décompression des images. Voici ces règles :

Utilisez des puissances de deux

En jeux vidéo, chaque bit compte ! C'est pourquoi les images qui servent de textures doivent toujours être à une dimension qui tient compte du monde binaire de l'informatique et qui remplit un espace mémoire (mesuré en octets) à la perfection. Les valeurs utilisables sont :

- **2** (utile si l'on veut une couleur simple)
- **4**
- **8**
- **16**
- **32**
- **64**
- **128** (utile pour les objets éloignés)
- **256** (idéal pour les appareils portables)
- **512** (idéal pour les objets secondaires)
- **1024** (idéal pour les consoles ou PC)
- **2048** (un peu lourd en mémoire)
- **4096** (idéal pour les consoles de jeux actuels en 2015)
- **8192** (PC haut de gammes avec des écrans 4K ou 8K)

Conseil!

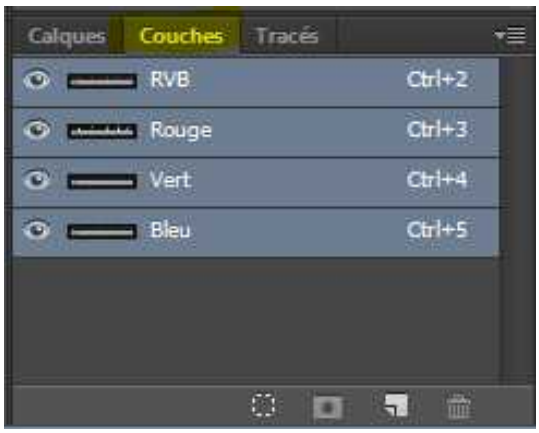
Les ***textures*** ne sont pas obligatoirement carrées, mais elles doivent respecter les puissances de deux!
Par exemple : **1024x512, 2048x128,** ou **2x4096** (idéal pour un dégradé horizontal)...

Notez-bien!

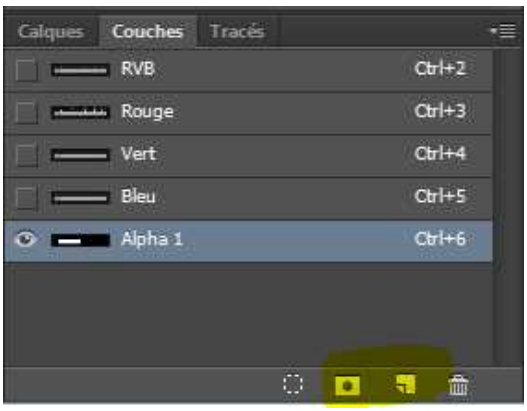
*Les seules exceptions à la règle des puissances de deux, sont pour les images de l'interface utilisateur (**IU** ou **UI** en anglais).*

Utilisez les couches dans Photoshop

Les textures dans le monde du jeu vidéo utilisent un système à trois couleurs (**RVB**) pour (**R**ouge, **V**ert et **B**leu). Photoshop permet de voir ces trois couches de couleurs dans l'onglet : ***Couches***.



Certains ***shaders*** vont utiliser ces trois couches de couleurs pour différentes fins. Comme exemple, la texture **Specular** utlise utilise les trois couches pour déterminer le pourcentage de réflectivité que chaque couleur émet.



Une quatrième couche (**A**) peut également être ajoutée en option afin de mettre de la transparence à l'image (**A**lpha).

Dans le bas du panneau *Couche*, il y a quatre icônes. Les deux icônes du milieu permettent d'ajouter une nouvelle **couche** à la liste, soit **à partir d'une sélection** ou une **couche vide**.

Exportez et importez vos images

Vous pouvez sauver vos images directement dans le répertoire **Assets** de votre projet. En ouvrant Unity, celui-ci détectera les ajouts.

Notez-bien!

*La couche Alpha peut servir à d'autres fins que la transparence. Par exemple, la texture Specular utilise l'Alpha pour définir le niveau de flou dans la réflexion (**Smoothness**).*

Unity est compatible nativement avec plusieurs formats d'images : (**.PSD, .TIFF, .JPG, .TGA, .PNG, .BMP, .IFF** et **.PICT**).

Notez-bien!

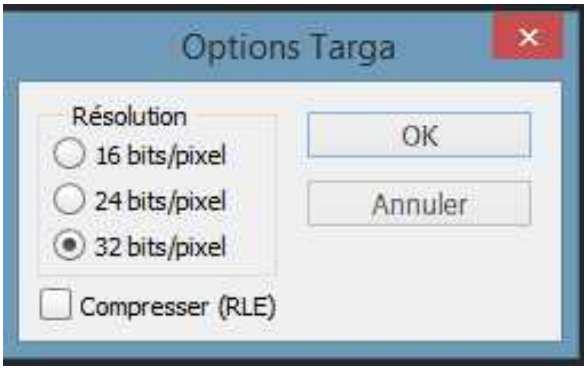
Si vous voulez conserver la couche Alpha dans votre image et conserver l'éditabilité de vos textures, utilisez le format natif de Photoshop (.PSD). Ce format sera converti automatiquement au démarrage d'Unity, mais il peut alourdir considérablement votre projet.

Conseil!

De tous les formats disponibles, je vous recommande personnellement d'utiliser **.TGA.**

Le format **.TGA** est non compressé et ses données sont brutes. Chaque pixel de couleur est individuellement inscrit dans la texture et ne requiert aucune modification de la part du processeur pour être lu.

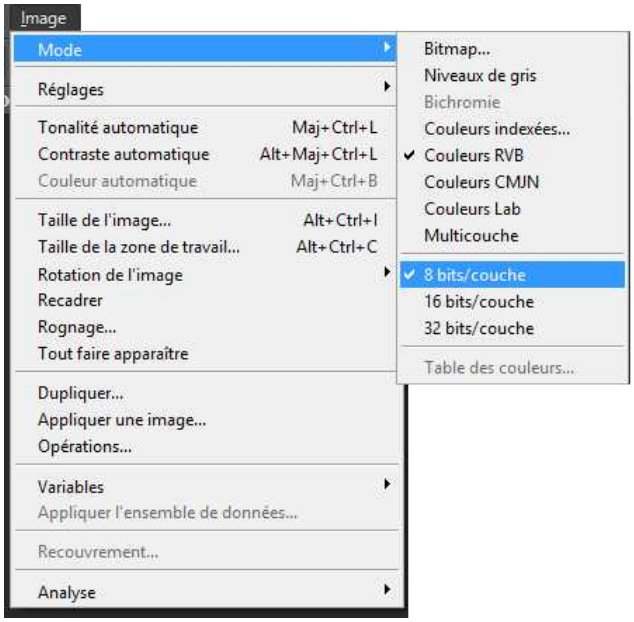
À l'exportation d'une image au format (**.TGA**), vous recevrez cette requête :



Si votre texture comprend une couche Alpha, prenez ***32 bits/pixel***, sinon ***16*** ou ***24 bits*** feront des fichiers beaucoup plus petits en mémoire *RAM* et en espace de stockage.

Notez-bien!

*Les images peuvent seulement être exportées au format (**.TGA**) si vous travaillez en mode **8 bit/couche** dans Photoshop.*



Conseil!

Il est déconseillé d'utiliser le format (**.JPG**) pour vos textures, car ce format ne contient pas de couche Alpha. Il utilise une compression destructive qui dégrade volontairement la qualité de l'image, dans le but de réduire la taille du fichier.

Je vous recommande donc d'utiliser le format (**.PNG**) pour les éléments de votre interface utilisateur. Il est relativement petit en taille de fichier et inclut l'Alpha directement dans la couleur, sans avoir de couches dédiées à la transparence.

Le processus de création des (GameObjects)

Voici maintenant l'explication des étapes de création d'un simple élément pour Unity. L'utilisation d'un objet 3d sera un bel exemple.

Créez votre objet

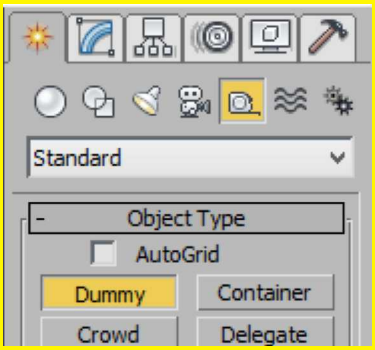
Avec un logiciel de création 3D : (**3DS Max, Maya, XSI, Lightwave, Cinema 4D, Cheetah 3D, Modo ou Blender**) créez un objet pour votre jeu. (Modélisation, textures et animations).

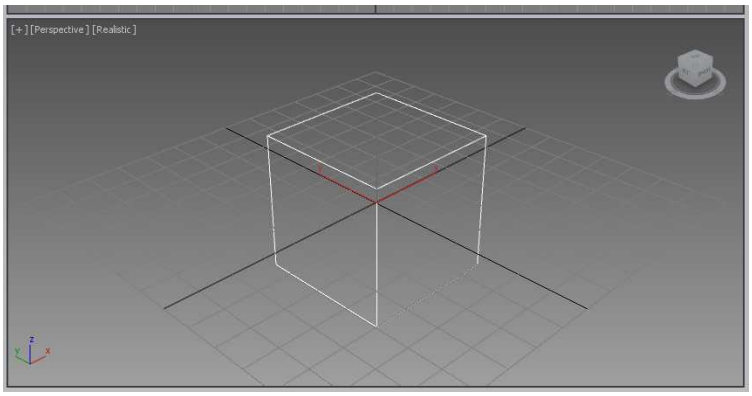
Notez-bien!

La création d'éléments 3D ne fait pas partie de la matière couverte dans ce livre. Vous pouvez trouver plusieurs manuels qui portent sur la modélisation 3D, chez votre libraire.

Conseil!

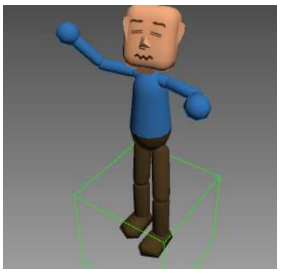
Voici une idée pratique, celle de débuter en créant un objet guide qui sera le parent de votre création.

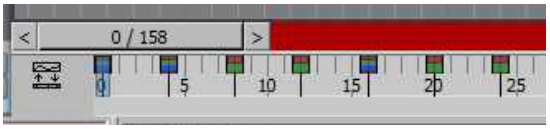




↘ Dans 3ds max un ***Dummy*** est parfait comme objet parent, car il n'a pas de *mesh* visuel.

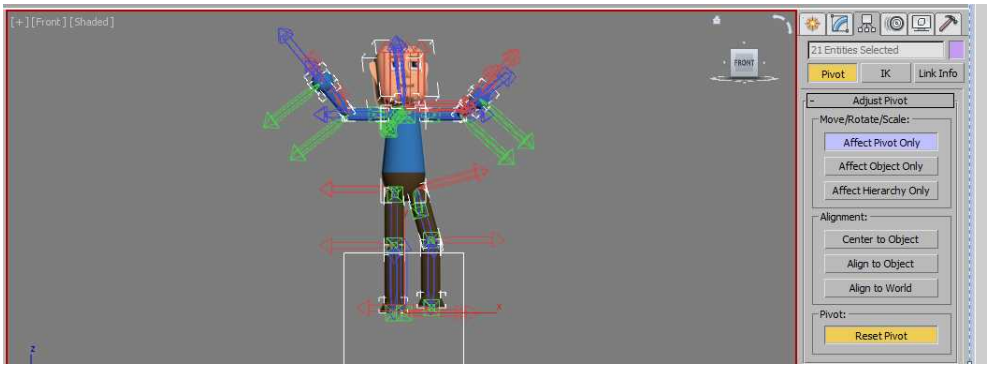
Une fois votre ***mesh*** créé, liez-le au ***Dummy*** afin de mettre ses coordonnées relatives à cet objet, plutôt qu'à la scène.

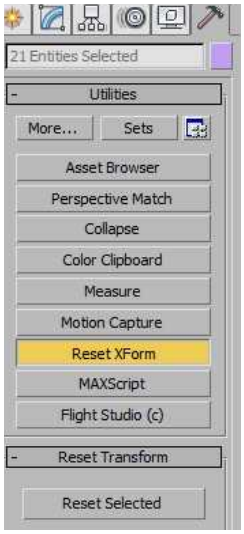




Avant d'exporter vos objets animés, vous devez mettre manuellement une image-clé sur votre ***mesh***, à l'image **0** et au début et à la fin de chaque animation sur la ligne du temps (***Timeline***), non pas sur le ***Dummy***, car celui-ci servira de point de références pour Unity.

Il est recommandé de réinitialiser les pivots de vos objets dans la scène. Surtout si ceux-ci ont été créés via une copie miroir, car les pivots sont inversés dans la copie.





Si vous avez redimensionné des objets dans la scène, il est possible qu'ils soient disproportionnés. Donc, il est recommandé de réinitialiser le ***XForm*** des objets avant de les exporter.

Exportez et importez vos objets

Vous pouvez sauver votre objet 3D directement dans le répertoire **Assets** de votre projet. En ouvrant Unity, celui-ci détectera les ajouts et importera ces objets en conséquence et ils se retrouveront dans le panneau *Project*. Unity utilise le format .**FBX** pour convertir vos modèles pour le jeu. Par contre, il est recommandé d'exporter vous-mêmes vos objets au format .**FBX** avant de les importer dans l'éditeur. À noter que le logiciel de modélisation doit être installé sur le poste de travail afin de réaliser la conversion.

Par exemple, si l'on crée un modèle 3D dans le logiciel 3DS Max et que l'on met un fichier au format .**MAX** dans le répertoire **Assets,** celui-ci sera converti automatiquement si 3DS Max est installé. Par contre, si l'on importe le projet sur un Mac, il sera impossible d'ouvrir le fichier .**MAX**, car le logiciel en question n'est pas disponible sur cette plateforme.

Notez-bien!

*Autodesk est le leader dans la création d'outils 3D, car celui-ci a racheté la majorité des compétiteurs de l'industrie. C'est pourquoi **Autocad**, **3DS Max**, **Maya** et **XSI** sont tous vendus par la même entreprise.*

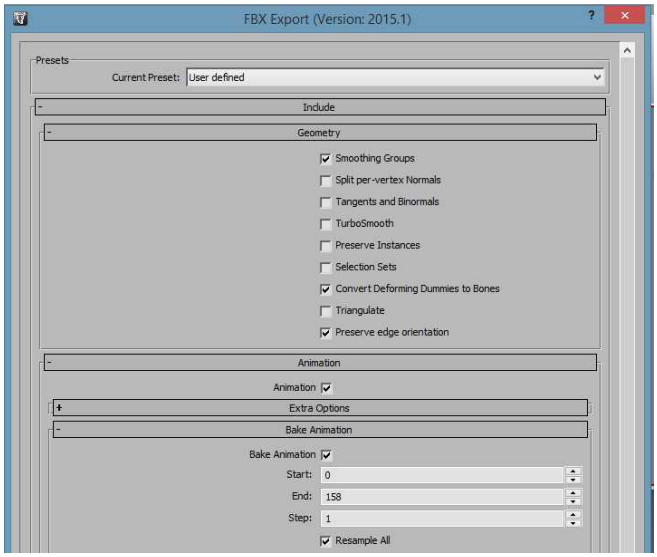
Unity peut comprendre nativement les formats 3D suivants: (**.FBX, .DAE** collada**, .DXF et .OBJ**)

Quand les logiciels de modélisation nécessaires sont installés sur le poste de travail, Unity peut également importer les formats (**.MAX, .MB, .MA, .BLENDER** et plusieurs autres).

Conseil!

De tous les formats disponibles, Unity recommande d'utiliser **.FBX**

Le format .**FBX** est supporté nativement dans tous les logiciels vendus par **Autodesk**, afin d'avoir un format universel qui contient toutes les propriétés importantes d'un modèle standardisé.

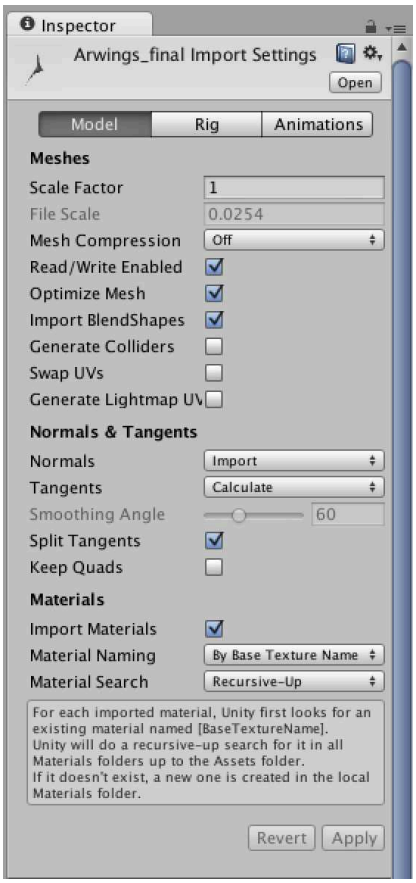


↘Dans les paramètres d'exportation, on laisse la majorité des paramètres par défaut intacte, mais on ajoute les options ***Smoothing Groups*** et ***Animation*** lorsque celles-ci sont utilisées.

Si vous avez des **animations**, cochez l'option ***Bake Animation*** et ***Resample All*** pour faciliter l'importation.

Vous pouvez exporter des fichiers **.FBX version 2012** et les versions plus récentes, mais il est recommandé de rester avec la version **2012** pour une meilleure compatibilité avec Unity.

Importez les paramètres de vos objets



↘Quand vous sélectionnez un élément dans le panneau *Project,* ses paramètres s'affichent dans *l'Inspector*. Les options sont différentes selon le type d'objet sélectionné.

Ajoutez vos objets dans la scène

Vous ajoutez un objet dans la scène en glissant celui-ci à partir du panneau *Project* vers le panneau *Hierarchy* ou directement dans la fenêtre *Scene*. Quand vous glissez un modèle 3D dans la scène, celui-ci sera ajouté automatiquement dans un ***GameObject*** avec un ***MeshRenderer***.

Assemblez les différents éléments

La relation entre les éléments les plus communs est :

• Une ***Texture*** est appliquée à un matériel ***Material***.

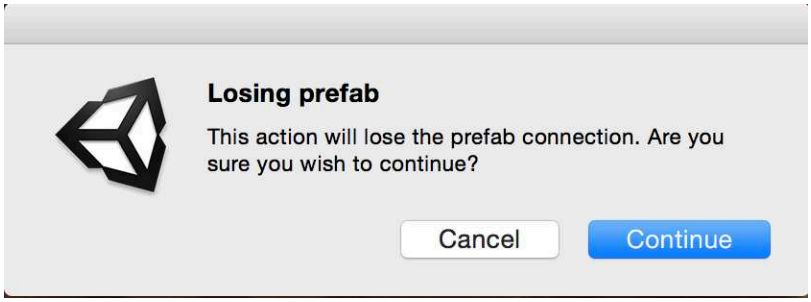
• Un ***Material*** est appliqué à un ***GameObect*** (avec le ***Component*** : ***MeshRenderer***).

• Une ***Animation*** est appliquée à un ***GameObject*** (avec le ***Component*** **:** ***Animation*** ou ***Animator***).

• Un ***Son*** est appliqué à un ***GameObject*** (avec le ***Component : AudioSource***).

Créez un Prefab

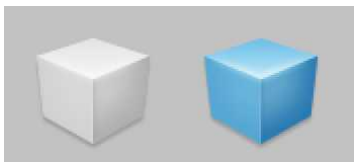
Par défaut, les changements qui sont appliqués au ***Prefab*** dans le panneau *Project,* sont automatiquement mis à jour dans la scène !



Par contre, si vous modifiez une instance d'un ***prefab*** dans la scène, la liaison avec le ***Prefab*** est alors brisée et les modifications ne sont pas appliquées aux autres instances, à moins que vous le spécifiez en appuyant sur le bouton « Apply » qui se trouve en-dessous du nom de l'objet dans *l'Inspector*.

↘ Cet avertissement vous informe que vous allez briser la liaison avec votre ***Prefab.***

↙ Ces boutons permettent de reconnecter le Prefab, soit en annulant (« **Revert** ») ou en appliquant les changements (« **Apply** »).



Vous pouvez créer un nouveau ***Prefab*** à partir d'un ***GameObject*** de la scène, simplement en le glissant du panneau *Hierarchy* vers *Project*. Vous pouvez également créer un nouveau ***Prefab*** vide et y glisser un contenu.

↘ **Prefab vide** à gauche et **plein** à droite.

Mise à jour des éléments

Lorsque vous voulez modifier vos objets importés, vous pouvez faire un double-clic sur un élément dans le panneau *Project*. L'application appropriée devrait s'ouvrir afin de vous permettre de mettre à

jour cet ***asset***. Lorsque vous avez terminé vos changements, sauvegardez les documents et Unity se chargera de mettre à jour vos modifications dans le jeu.

Création de scènes

Les scènes contiennent les éléments de votre jeu. On utilise les scènes pour les menus, les niveaux ainsi que tout le reste. Considérez les scènes comme différents niveaux dans un jeu. Dans chacune d'elle, placez les objets qui constituent vos environnements, obstacles, ennemis et alliés. Le jeu se déroule évidemment dans les scènes.

Ajoutez des *Components* et des *scripts*

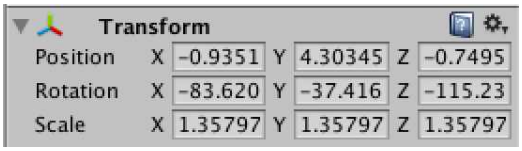
Dans votre scène, vous pouvez ajouter de nouveaux ***Components*** aux éléments. Tout ce que vous devez faire, c'est de sélectionner un ***GameObject*** ou un ***Prefab*** dans le panneau *Hierarchy*, puis dans la barre de menus, allez dans : **Component**.

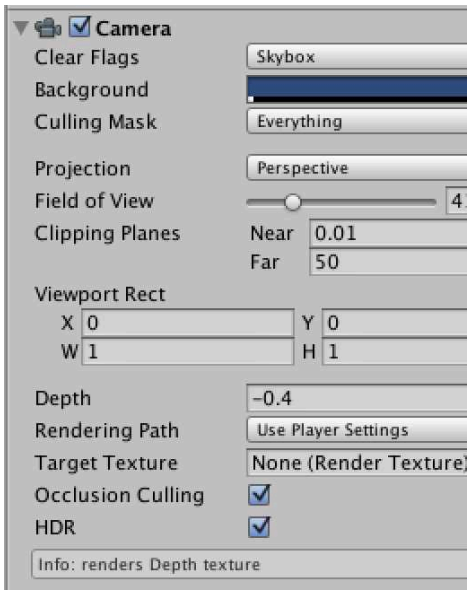
De là, vous obtiendrez une liste de tous les ***Components*** disponibles. Il y a également le bouton « Add Component » dans *l'Inspector* vous permettant d'accomplir la même tâche.

Faites de même pour les scripts. Vous pouvez également ajouter un script à un ***GameObject*** simplement en le glissant sur celui-ci.

Placez vos *GameObjects*

Lorsque vos objets sont placés dans la scène, vous pouvez changer leur position, rotation et proportion, en utilisant l'outil ***Transform,*** qui se trouve dans *l'Inspector*. Vous pouvez ainsi effectuer les derniers ajustements.





Travaillez avec des (*Cameras*)

Les caméras sont les yeux de votre jeu. Tout ce que le joueur voit durant le jeu se fait à l'aide d'une ou de plusieurs caméras. Vous pouvez positionner et aligner les caméras comme vous le faites pour tous les autres ***GameObjects***. Une caméra est simplement un ***GameObject*** avec le ***Component*** (***Camera***) attaché à celui-ci. C'est pourquoi une caméra peut faire tout ce qu'un ***GameObject*** peut accomplir, en plus des tâches propres à celle-ci. Vous verrez dans les prochains chapitres comment agencer les caméras pour satisfaire différents besoins.

Ajustez l'éclairage (*Lights*)

À quelques exceptions près, vous devez toujours utiliser au moins une lumière (***Light***) dans vos scènes.

Il existe trois types de lumière et elles se comportent différemment. Il est important de se rappeler qu'elles créent l'atmosphère et l'ambiance dans votre jeu. Expérimentez avec les lumières afin de comprendre comment celles-ci fonctionnent. Par défaut, les scènes contiennent une ***Directional Light***.

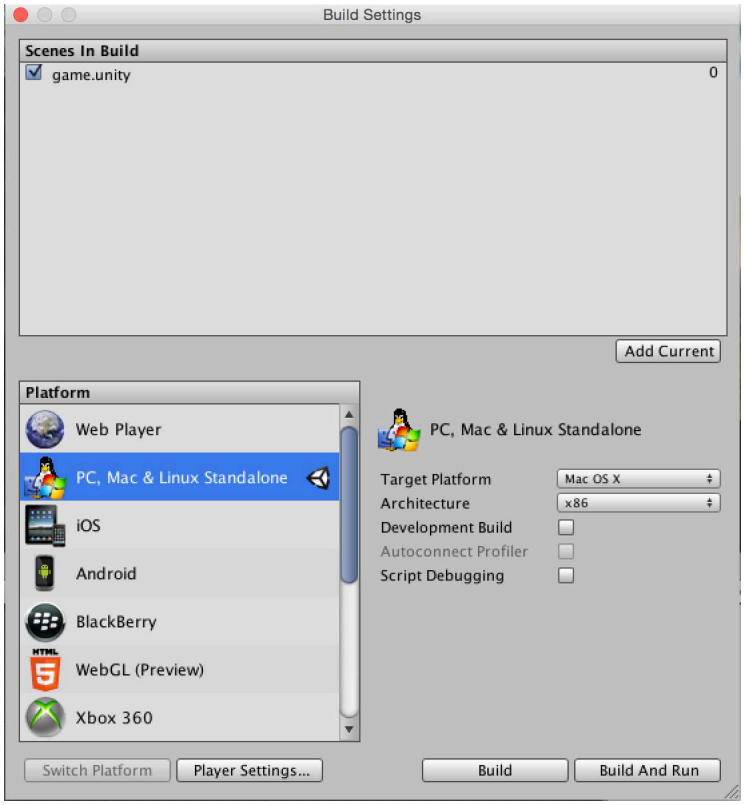
Publiez vos projets (Build)

À tout moment, lors de votre développement, il est possible que vous vouliez tester votre jeu dans une application ou une page Web indépendante de l'éditeur. Cette section explique comment accéder aux paramètres de publications (***Build Settings***) et comment créer différentes versions de vos jeux.

On accède à ces paramètres, à partir de la barre de menus dans :

File | Build Settings...

Raccourcis : (⇧+CTRL+B) ou **(⇧+⌘+B)** sur Mac**.**



Ce menu contient la liste des **scènes** incluses dans le (***Build***) de votre jeu.

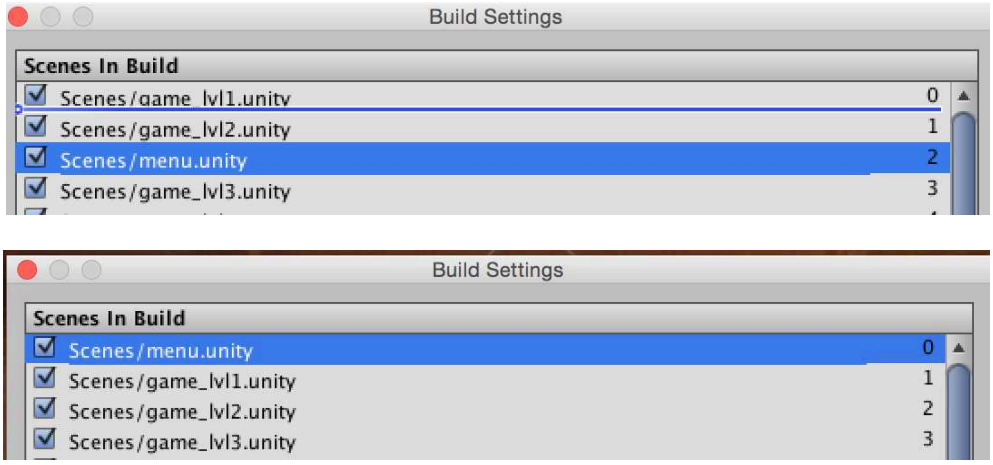
La première fois que vous ouvrez cette fenêtre, à partir d'un nouveau projet, cette liste sera vide. Si vous exportez votre jeu à ce moment précis, seulement la scène courante sera incluse à l'exportation. Pour faire un test rapide, laissez cette liste vide et exportez votre projet avec le bouton « Build ».

Il est facile d'ajouter des scènes à cette liste, il y a deux façons d'y arriver.

• La première est d'appuyer sur le bouton « Add Current » pour y ajouter la scène active.

• La seconde méthode est de glisser celle-ci du panneau *Project*, vers cette liste.

À ce moment, vous devriez remarquer que les scènes ont une valeur d'index différente. La scène (**0**) est la première à être chargée à l'exécution du jeu. Quand vous voulez charger une nouvelle scène, utilisez la commande : **Application.LoadLevel()** dans votre script.



Vous pouvez réorganiser vos scènes simplement en les glissant dans l'ordre désiré.

Vous pouvez aussi retirer une scène de la liste avec la touche (**Supprim.**) sur Windows ou (**⌘+Retour Arrière**) sur Mac. La scène ne sera pas incluse à l'exportation, mais elle reste présente dans le projet.

Quand vous êtes prêts à publier un ***Build***, sélectionnez la plateforme que vous voulez utiliser parmi la liste. Si le logo d'Unity n'est pas présent à côté de la plateforme, appuyez sur « Switch Platform » pour changer de système.

En appuyant sur le bouton « Build » ou « Build And Run » vous serez en mesure d'exporter une version de votre jeu, en quelques minutes. Faire un jeu avec Unity, est donc très simple.

Finalement, si vous voulez tester votre application en étant assisté par un débogueur, cochez la case « Development Build », l'exécution de vos scripts sera accessible durant celle-ci afin de détecter les problèmes potentiels.

Diffusez avec Web Player (*Streamed*)

La diffusion en continu (***Streaming***) permet à une version ***Web*** du jeu de se lancer aussitôt que la scène (**0**) est complètement chargée. Si votre jeu contient 10 niveaux, il est inconcevable de penser qu'un joueur devra attendre que tous les niveaux soient chargés avant de lancer l'application. Il est donc important que la scène que vous voulez charger soit d'abord complètement téléchargée avant de la lancer.

Le (***Web Player***) va démarrer aussitôt que tous les éléments de la scène (**0**) sont téléchargés. En d'autres mots, le jeu va se lancer plus rapidement si vous utilisez le paramètre de diffusion (***Streamed***) pour votre projet.

Notez-bien!

L'action importante à retenir, c'est que l'on doit s'assurer que la scène est prête avant de la lancer.

Normalement, lorsqu'un jeu n'est pas en diffusion (***Streamed***), on utilise cette commande pour changer de scène.

```
Application.LoadLevel("NomDuNiveau");
```

Dans un jeu en diffusion, vous devriez toujours valider que la scène soit chargée avec la fonction **CanStreamedLevelBeLoaded().** Voilà à quoi le code devrait ressembler.

```
var niveauACharger = 1;

function loadNewLevel(){
   if (Application.CanStreamedBeLoaded (niveauACharger)){
      Application.LoadLevel(niveauACharger);
   }
}
```

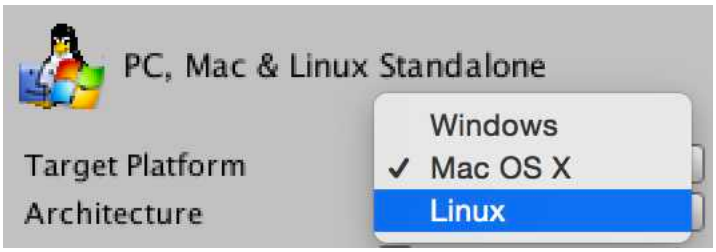
Si vous voulez afficher le pourcentage du téléchargement complété, vous pouvez récupérer la valeur avec la fonction : **GetStreamProgressForLevel().**

Déployez hors ligne (Offline Web Player)

Quand l'option de déploiement hors ligne (***Offline Deployment***) est activée pour la plateforme (***Web Player***), le fichier **UnityObject.js** (qui est utilisé pour l'interface entre l'utilisateur et le serveur) sera inclus à l'exportation, permettant à votre jeu de fonctionner, même lorsqu'il n'y a pas de connexion Internet.

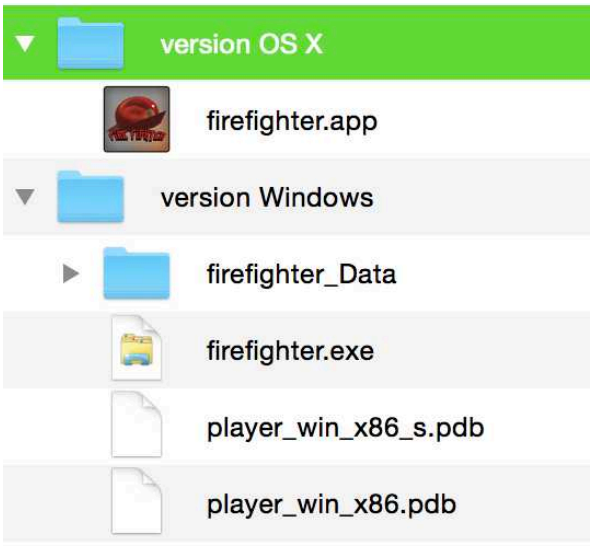
Autrement, le fichier **UnityObject.js** sera téléchargé directement du serveur d'Unity afin de toujours utiliser la dernière version.

Déployez une application redistribuable (*Standalone*)



Avec Unity, vous pouvez compiler une version de votre jeu pour Windows, Mac et Linux. Il suffit d'indiquer dans quelle plateforme vous voulez publier votre application et d'appuyer sur le bouton « Build ».

Selon la plateforme choisie, les fichiers résultants changeront. Sur Windows, vous obtiendrez un fichier exécutable (**.exe**) ainsi qu'un répertoire de données (***_Data***) avec les ressources nécessaires. Sur Mac, une application sera exportée (**.app**) et toutes les données seront incluses dans ce fichier, il en va de même pour Linux.



Pour la distribution sur Mac, vous n'avez qu'à utiliser l'application exportée, tout est inclus. Sur Windows, vous devez absolument fournir le répertoire (***_Data***) avec votre exécutable, sinon, le jeu ne fonctionnera pas.

En détails...

Le processus d'exportation va d'abord placer un exécutable vide à l'emplacement spécifié. Unity va ensuite passer au travers la liste des scènes marquées pour l'exportation pour ensuite les ouvrir dans l'éditeur une à la fois, les optimiser et les intégrer dans l'exportation. Unity va également recenser tous les éléments requis par les différentes scènes et créera un fichier de data séparé dans le paquet exporté.

• Tous les **GameObjects** avec un marqueur (**Tag**) : **EditorOnly** ne seront pas inclus à l'exportation.

• Quand un niveau récent est chargé, tous les éléments de la scène précédente sont effacés. Vous pouvez prévenir cette destruction en y ajoutant la fonction : **DontDestroyOnLoad()** sur les objets que vous voulez conserver entre les scènes. Vous pouvez ainsi permettre à une musique de continuer à jouer entre les scènes, ou à un contrôleur de garder une trace des progrès du joueur, entre celles-ci.

• Lorsque tous les éléments d'une scène sont chargés, le message : **OnLevelWasLoaded()** sera envoyé à tous les **GameObject** actifs.

Chargement au préalable

Un programme redistribuable chargera préalablement et automatiquement tous les éléments d'une scène, durant son exécution.

La seule exception est la scène (**0**), car les éditeurs y placent généralement les informations légales, l'affichage doit se faire le plus rapidement possible.

Si vous voulez être certains que tout le contenu est chargé au préalable, vous pouvez créer une scène vide qui utilisera un script, avec la commande : **Application.LoadLevel(1).** Ensuite, placez cette scène vide à l'index (0), dans la liste d'exportation. De cette façon, tous les niveaux suivants seront déjà chargés.

Vous êtes maintenant prêts à publier vos jeux!

Les raccourcis du logiciel

Voici une liste des raccourcis, par défaut, du logiciel.

Raccourcis : Outils	
Q	Translation (Pan) de la caméra
W	Déplacement
E	Rotation
R	Redimensionner
T	Outil (Rect Tool)
Z	Change le mode du pivot
X	Change la rotation du pivot
V	Active le magnétisme (snap) sur les Vertex.
CTRL+Clic (Win) ⌘+✳(Mac)	Active le magnétisme (snap)

Raccourcis : GameObject	
CTRL+⇧+N (Win) ⌘+⇧+N (Mac)	Crée un nouveau GameObject
CTRL+ALT+F (Win) ⌘+Opt.+F (Mac)	Déplace le GameObject devant la caméra
CTRL+⇧+F (Win) ⌘+⇧.+F (Mac)	Aligne le GameObject selon la vue de la caméra
⇧+F ou F (deux fois)	Verrouille la caméra de la fenêtre (Scene) sur l'objet sélectionné.

Raccourcis : Assets	
CTRL+R (Win) ⌘+R (Mac)	Rafraîchir

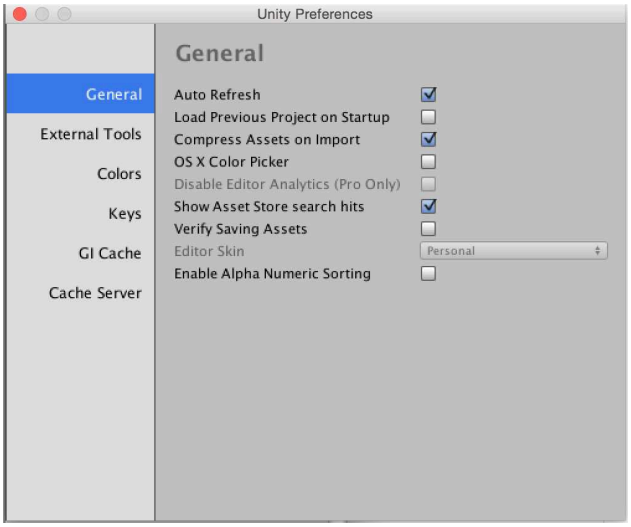
Raccourcis : Fenêtre	
CTRL+1 (Win) ⌘+1 (Mac)	Scene
CTRL+2 (Win) ⌘+2 (Mac)	Game
CTRL+3 (Win) ⌘+3 (Mac)	Inspector
CTRL+4 (Win) ⌘+4 (Mac)	Hierarchy
CTRL+5 (Win) ⌘+5 (Mac)	Project
CTRL+6 (Win) ⌘+6 (Mac)	Animation
CTRL+7 (Win) ⌘+7 (Mac)	Profiler
CTRL+9 (Win) ⌘+9 (Mac)	Magasin en ligne (Asset store)
CTRL+0 (Win) ⌘+0 (Mac)	Version Control
CTRL+⇧+C (Win) ⌘+⇧+C (Mac)	Console

Raccourcis : Edit	
CTRL+Z (Win) ⌘+Z (Mac)	Annuler
CTRL+Y (Win) ⌘+⇧+Z (Mac)	Rétablir
CTRL+X (Win) ⌘+X (Mac)	Couper
CTRL+C (Win) ⌘+C (Mac)	Copier
CTRL+V (Win) ⌘+V (Mac)	Coller
CTRL+D (Win) ⌘+D (Mac)	Dupliquer
⇧+Supprim.	Supprimer

Raccourcis : Edit	
F	Centrer sur la sélection
CTRL+F (Win) ⌘+F (Mac)	Rechercher
CTRL+A (Win) ⌘+A (Mac)	Sélectionner tout
CTRL+P (Win) ⌘+P (Mac)	Jouer (Play)
CTRL+⇧+P (Win) ⌘+⇧+P (Mac)	Pause
CTRL+ALT+P (Win) ⌘+⌥+P (Mac)	Avancer par étape (Step)

Raccourcis : Sélection	
CTRL+⇧+1 (Win) ⌘+⇧+1 (Mac)	Charger la sélection 1
CTRL+⇧+2 (Win) ⌘+⇧+2 (Mac)	Charger la sélection 2
CTRL+⇧+3 (Win) ⌘+⇧+3 (Mac)	Charger la sélection 3
CTRL+⇧+4 (Win) ⌘+⇧+4 (Mac)	Charger la sélection 4
CTRL+⇧+5 (Win) ⌘+⇧+5 (Mac)	Charger la sélection 5
CTRL+⇧+6 (Win) ⌘+⇧+6 (Mac)	Charger la sélection 6
CTRL+⇧+7 (Win) ⌘+⇧+7 (Mac)	Charger la sélection 7
CTRL+⇧+8 (Win) ⌘+⇧+8 (Mac)	Charger la sélection 8
CTRL+⇧+9 (Win) ⌘+⇧+9 (Mac)	Charger la sélection 9
CTRL+ALT+1 (Win) ⌘+⌥.+1 (Mac)	Sauver la sélection 1
CTRL+ALT+2 (Win) ⌘+⌥.+2 (Mac)	Sauver la sélection 2

Raccourcis : Sélection	
CTRL+ALT+3 (Win) ⌘+⌥.+3 (Mac)	Sauver la sélection 3
CTRL+ALT+4 (Win) ⌘+⌥.+4 (Mac)	Sauver la sélection 4
CTRL+ALT+5 (Win) ⌘+⌥.+5 (Mac)	Sauver la sélection 5
CTRL+ALT+6 (Win) ⌘+⌥.+6 (Mac)	Sauver la sélection 6
CTRL+ALT+7 (Win) ⌘+⌥.+7 (Mac)	Sauver la sélection 7
CTRL+ALT+8 (Win) ⌘+⌥.+8 (Mac)	Sauver la sélection 8
CTRL+ALT+9 (Win) ⌘+⌥.+9 (Mac)	Sauver la sélection 9



Les préférences (Preferences)

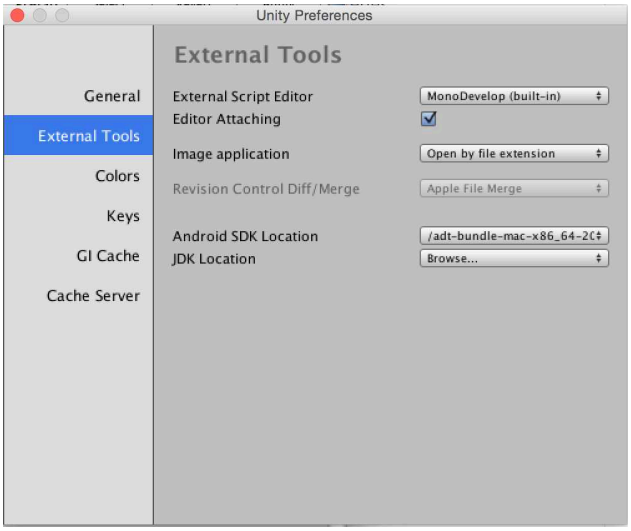
Unity fournit plusieurs options pour personnaliser le comportement de l'éditeur.

Dans la barre de menus, allez à :

Edit | Preferences…

Générales (General)

Paramètres	Description
Auto Refresh	Est-ce que l'éditeur doit mettre les fichiers à jour automatiquement quand un changement est détecté ?
Load Previous Project on Startup	Au démarrage, est-ce que l'on doit reprendre là où l'on a quitté ?
Compress Assets On Import	Est-ce que les éléments doivent être compressés automatiquement durant l'importation ?
OSX Color Picker	(Mac) Doit-on utiliser les outils de sélection de couleurs de OS X au lieu de l'outil d'Unity ?
Disable Editor Analytics (Pro Only)	Est-ce que l'éditeur doit arrêter d'envoyer des informations à Unity automatiquement ?
Show Asset Store search hits	Doit-on montrer le nombre d'éléments disponibles dans le magasin en ligne, sur le panneau Project ?
Verify Saving Assets	Est-ce qu'Unity doit vérifier quels sont les éléments à sauver lorsqu'on quitte ?
Editor Skin	Quelle couleur utiliser pour l'interface. (Unity Pro utilise le thème « Dark » par défaut)
Enable Alpha Numeric Sorting	Doit-on activer le tri des GameObject en ordre alphabétique ?



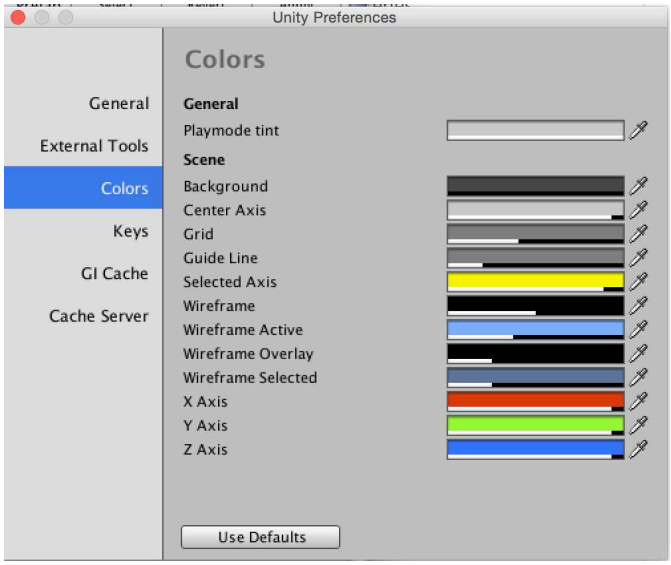
Outils externes (External Tools)

Paramètres	Description
External Script Editor	Quelle application Unity doit utiliser pour éditer les scripts ?
External Script Editor Args	(Win) Quels sont les arguments à passer à l'éditeur externe ? $(File) sera remplacé par l'emplacement du fichier. $(Line) sera remplacé par le numéro de ligne que l'éditeur doit se rendre. (Voir les exemples ci-dessous)
Editor Attaching	Est-ce qu'Unity permet à l'éditeur externe de contrôler le débogage ?
Image application	Quelle application Unity doit utiliser pour ouvrir une image ?
Revision Control Diff/Merge	Quelle application Unity doit utiliser pour résoudre les conflits de versions venant du serveur ?
Android SDK Location	Quel est l'emplacement du SDK d'Android sur votre poste de travail ?
JDK Location	Quel est l'emplacement de Java JDK sur votre poste de travail ?

Exemples d'arguments pour l'éditeur de script :

Gvim/Vim : --remote-tab-silent +$(Line) "$File"
Notepad2 : –g $(Line) "$(File)"
Sublime Text 2 : "$File" :$(Line)
Notepad++ : -n$(Line) "$(File)"

Couleurs (Colors)

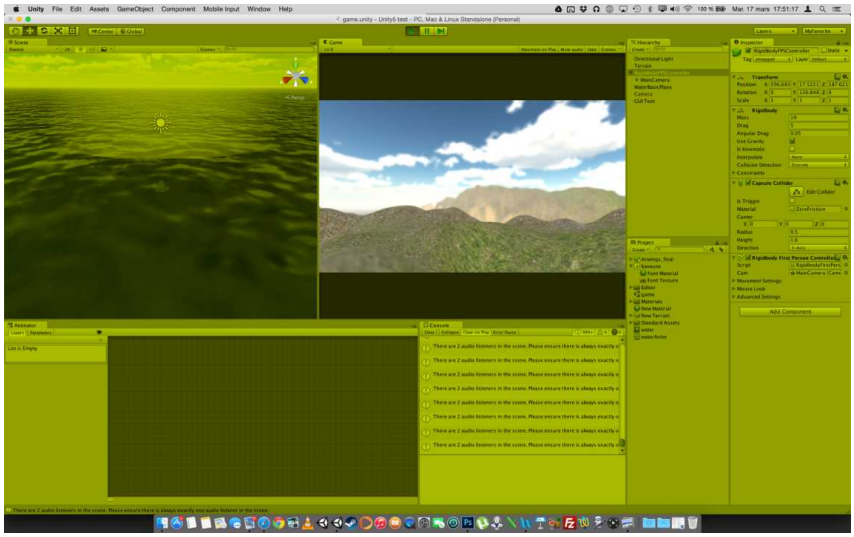


↘ Ce panneau vous permet de changer les couleurs utilisées par Unity pour les différents éléments de l'interface.

Conseil!

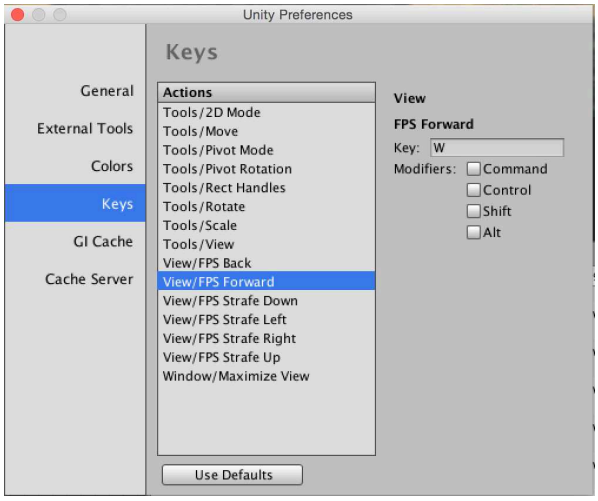
Une notion importante à retenir, quand on développe avec Unity, c'est que les changements que vous effectuez lorsque l'éditeur est en mode (***Play***), ne sont pas sauvegardés.

C'est pourquoi, nous vous recommandons de changer la couleur (***Playmode tint***) pour une couleur distincte, car cela indique clairement quand vous ne pouvez pas faire de changements.



↘ Interface en mode (*Play*) avec une couleur distincte.

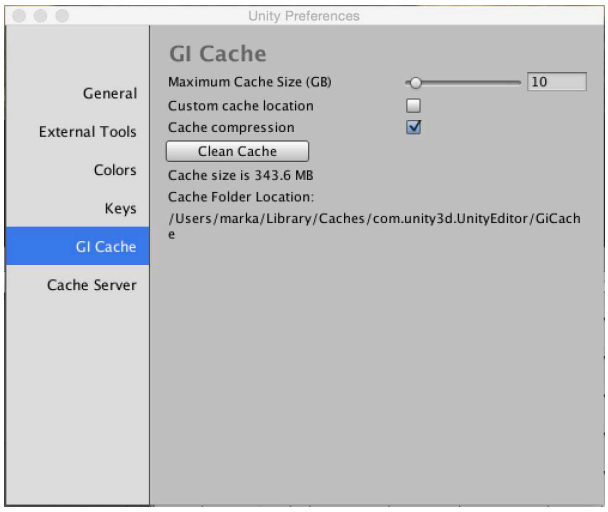
Les touches clés (Keys)



↘Ce panneau permet de changer les différents raccourcis clavier pour Unity.

Conseil!

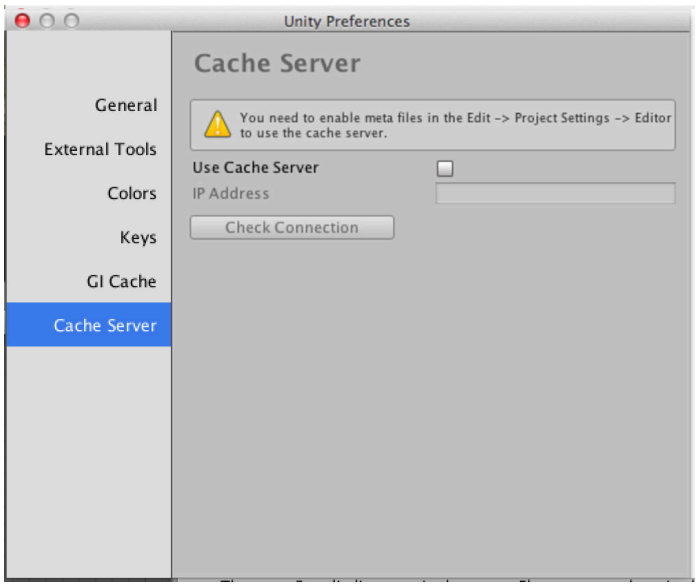
Si vous utilisez un clavier **AZERTY,** c'est à ce moment que vous avez la possibilité de modifier les contrôles, pour la disposition du clavier.



Mémoire cache pour l'illumination globale (GI Cache)

↘Ce panneau permet d'ajuster l'espace de disque alloué pour emmagasiner des données relatives à l'illumination globale.

Paramètres	Description
Maximum Cache Size (GB)	Taille allouée sur le disque en giga-octets
Custom cache location	Doit-on utiliser en emplacement personnalisé pour le cache ? (utile si le disque principal est très petit)
Cache compression	Doit-on compresser les informations du cache ?
Clean Cache	Permet de supprimer les données emmagasinées.
Cache size is	Indique la taille du cache actuellement utilisée.
Cache Folder Location :	Emplacement sur le disque pour le cache.



Serveur de mémoire cache (Cache Server)

Paramètres	Description
Use Cache Server	Doit-on activer le serveur cache ?
IP Adress	Adresse IP du serveur cache, si actif.

Vous voici rendus à la fin du premier chapitre de cet ouvrage. Si la théorie vous a paru un peu compliquée, ne vous en faites pas, ces outils deviendront une seconde nature après quelques exercices. Avant de s'aventurer plus loin dans le logiciel, arrêtez-vous afin de vous familiariser avec quelques fondements du **game design.** Ceux-ci permettront de mieux comprendre et planifier vos futures productions car tout se passe avant celles-ci, durant la planification.

Chapitre 2 - Jeu *#1 : Rebonds*

Matière couverte dans ce chapitre

- **Le concept du jeu**
- **L'utilisation des modèles de bases**
- **L'utilisation de modèles importés (.FBX)**
- **L'utilisation de la physique**
- **La création d'un radar**
- **La création d'une interface utilisateur simple**

Il est maintenant temps de mettre en pratique certaines notions apprises et de se lancer tête première dans Unity® afin de créer notre premier prototype.

Êtes-vous prêts ?

Voici ce que nous allons faire :

Premièrement, nous allons élaborer un concept de jeu. Ensuite, nous allons créer une scène avec des modules de formes simples fournis par Unity. Nous allons ajuster l'éclairage et l'ambiance de l'univers pour ensuite créer des effets afin d'embellir notre jeu. Par la suite, nous allons activer la physique ***PhysX*** qui est incluse avec le logiciel, ajouter ensuite un ***physic Material*** afin de définir les propriétés de nos objets. Nous allons terminer le chapitre avec les interfaces utilisateurs (***IU*** ou ***UI*** en anglais), en créant un guide visuel de la scène sous forme de radar ou de carte, ainsi que le pointage du jeu, en texte.

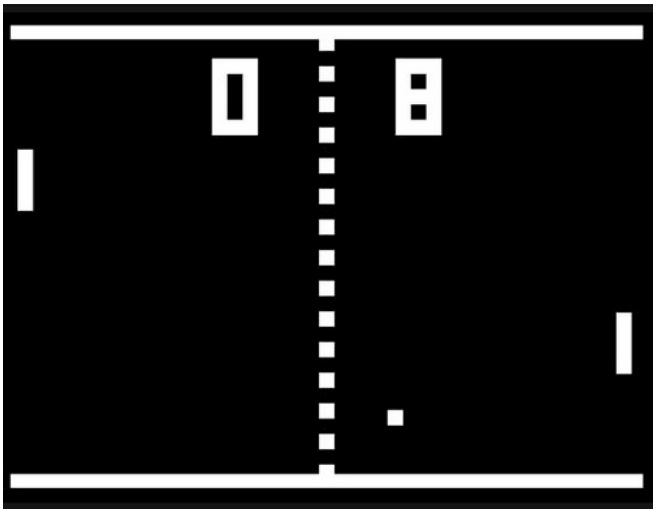
Débutons !

Le concept

Comme c'est un premier projet, nous allons nous concentrer sur un gameplay simple, que nous pourrons accomplir en peu de temps avec un minimum de connaissances en création 3D et en programmation.

Exemple : Pong

Pong est un jeu très simple développé par *Allan Alcorn* durant ses débuts chez *Atari*. Les instructions pour le jeu étaient :



1. ***Deposit Quarter*** (Dépose une pièce).

2. ***Ball Will Serve Automatically*** (La balle sera servie automatiquement).

3. ***Avoid Missing Ball For High Score*** (Évite de manquer la balle pour un haut pointage).

Nous allons utiliser les **2 dernières instructions** pour notre jeu, mais avec un seul joueur et dans un environnement 3D.

Le gameplay :

1. **Structure**: ***arcade*** (rythme rapide pour atteindre et battre son record).

1.1. **Catégorie** : ***déplacement à la troisième personne*** (on contrôle une palette).

1.1.1. **Traitement** : ***vitesse*** (Plus le jeu progresse, plus celui-ci s'intensifie).

La clentèle :

2. **Sexe** : ***mâle*** (Aspect compétitif)

2.1. **Âge** : ***18-28*** (Fin de l'adolescence, jeunes adultes et bons réflexes)

2.1.1. **Personnalité** : ***perfectionniste*** (Intense et mettant l'emphase sur le **gameplay**).

Le genre:

Le genre : ***action,*** le **sous-genre *: balle et palette***

En tenant compte de ces éléments, nous sommes prêts à travailler.

Avant de débuter

Vous aurez besoin du paquet : **Chapitre2-4.unitypackage** qui contient tous les éléments graphiques de l'exercice.

Atelier pratique

Mise en place des éléments du gameplay

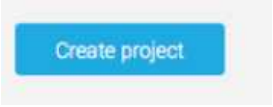
1. Ouvrez Unity®.

2. Cliquez sur le bouton : « New Project ».

3. Sauvez votre projet sur un disque dur local en inscrivant le nom: **Rebonds.**

3.1. Ce sera un projet : **3D.**

3.2. Appuyez sur « Create Project ».



Lorsqu'Unity est ouvert :

4. **Sauvegardez votre scène** (**CTRL+S**) ou (**⌘+S**) sur Mac.

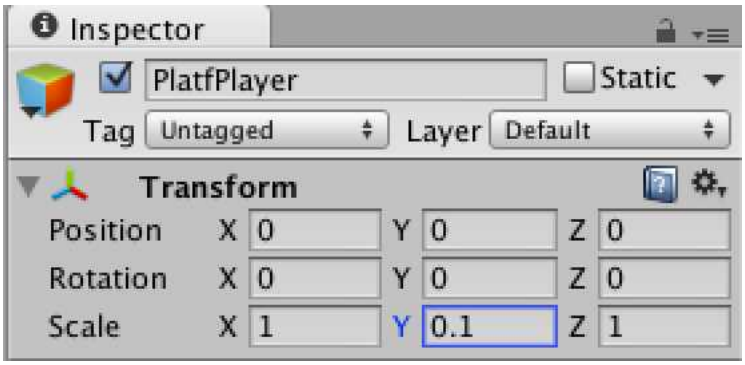
4.1. Nommez la scène : **Game**

5. Maintenant, nous allons créer une palette contrôlée par le joueur et utiliser un cube que l'on va amincir.

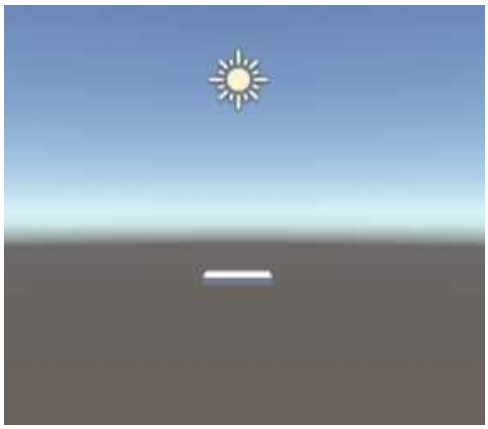
5.1. Dans la barre de menus, cliquez sur :

GameObject | 3D Object | Cube

5.2. Dans le panneau *Inspector*, changez les paramètres :



Transform
5.2.1. ***Nom*** **:** PlatfPlayer
5.2.2. ***Position*** **: x**=0**, y**=0**, z**=0
5.2.3. ***Scale*** **: x**=1**, y**=0.1**, z**=1

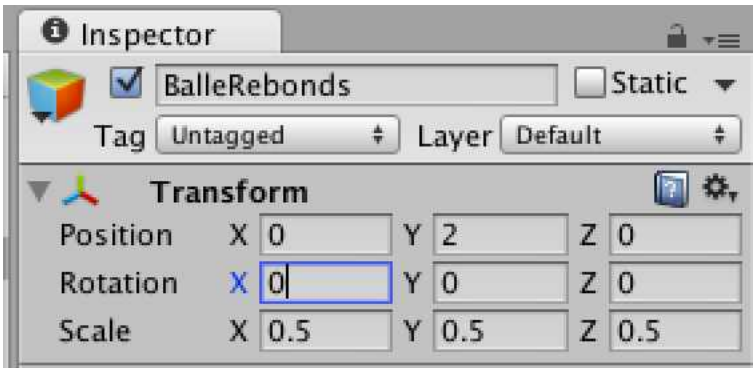


6. Ensuite, il faut créer une balle pour le défi.

6.1. Dans la barre de menus, cliquez sur :

GameObject | 3D Object | Sphere

6.2. Dans le panneau *Inspector*, changez les paramètres :

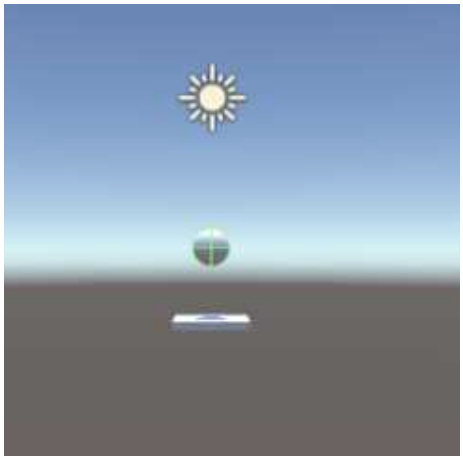


Transform

6.2.1. ***Nom*** : BalleRebonds

6.2.2. ***Position*** **: x**=0, **y**=2, **z**=0

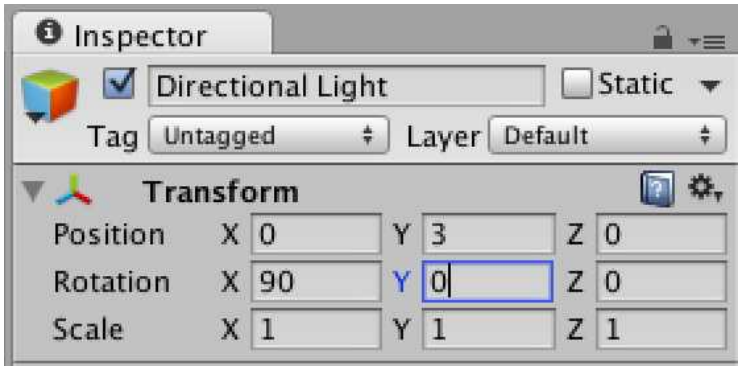
6.2.3. ***Scale*** **: x**=0.5, **y**=0.5, **z**=0.5



7. Pour guider le joueur vers la balle, nous allons projeter une ombre vers le bas.

7.1. Dans le panneau *Hierarchy*, sélectionnez la lumière : **Directional Light.**

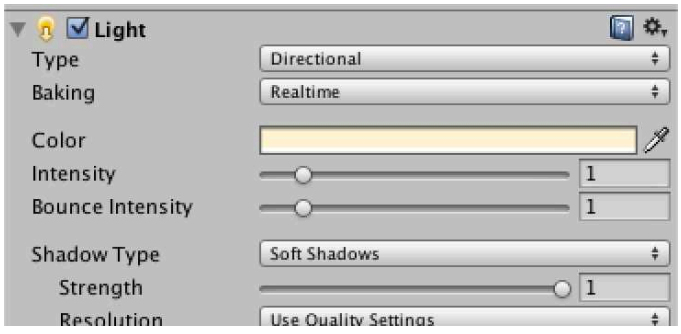
7.2. Dans l'onglet *Inspector*, changez les paramètres :



Transform

7.2.1. ***Rotation*** : **x**=90, **y**=0, **z**=0

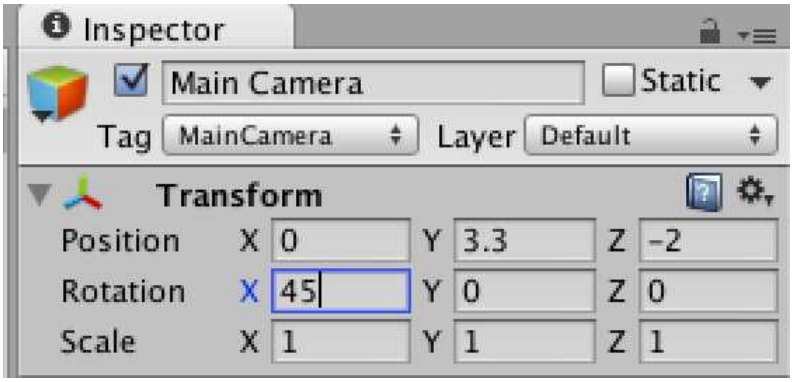
7.3. Assurez-vous que l'option : **Shadow Type** est bel et bien activé avec ***Soft Shadows*** ou ***Hard Shadows***.



8. Vous devez maintenant replacer la caméra dans la scène pour bien percevoir l'action du jeu !

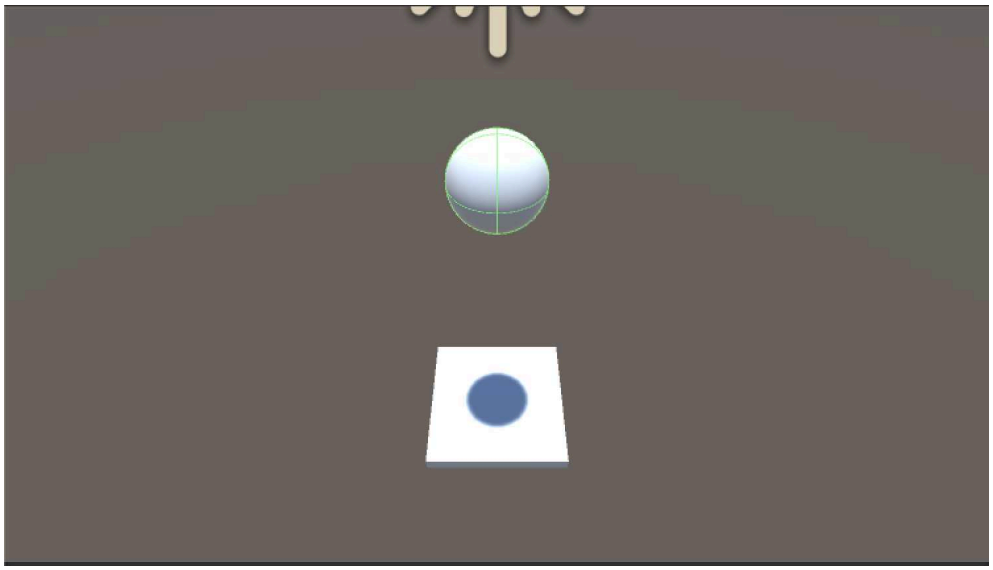
8.1. Dans le panneau *Hierarchy*, sélectionnez : **Main Camera.**

8.2. Dans *l'Inspector*, changez :



Transform
8.2.1. ***Position*** : **x**=0, **y**=3.3, **z**=-2
8.2.2. ***Rotation*** **:x**=45, **y**=0, **z**=0

Dans la scène, vous devriez voir la balle et la palette ainsi que son ombre, dans la fenêtre *Game*.



Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac!

Pour la prochaine étape, vous allez importer le paquet **Chapitre2-4.unitypackage** qui renferme nos ***assets*** pour le jeu (sons, textures et modèle 3D).

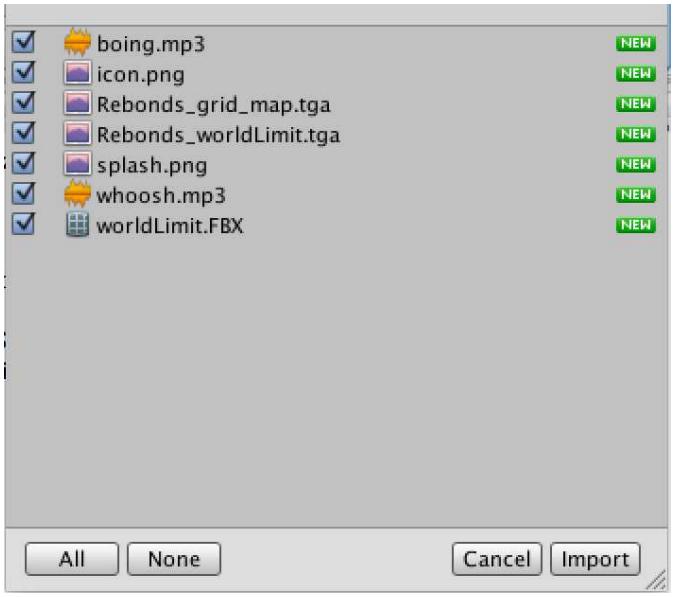
9. En passant par la barre de menus, faites :

Assets | Import Package | Custom Package…

Vous pouvez faire également un clic secondaire au-dessus du panneau *Project,* puis :

➢ **Import Package > Custom Package…**

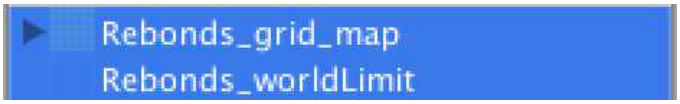
Les deux méthodes sont identiques.



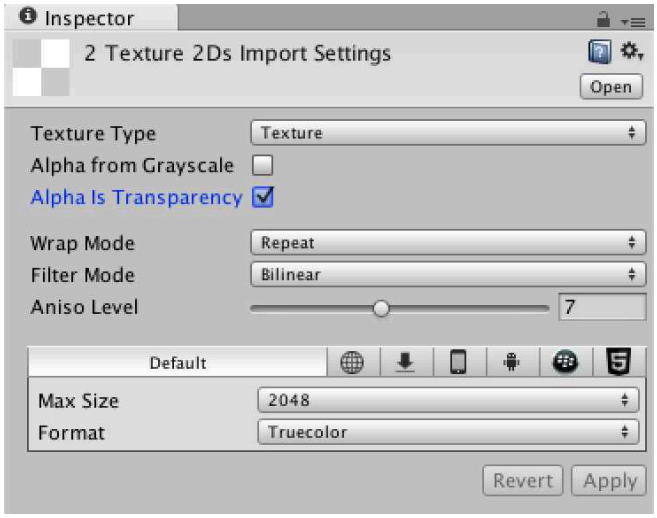
Sélectionnez le paquet **Chapitre2-4.unitypackage** depuis les sources du livre et importez son contenu.

Maintenant, vous devriez avoir **Rebonds_worldLimit.tga** : un contour lumineux afin de délimiter la frontière du jeu. **Rebonds_worldlimit.fbx :** une zone limite pour le gameplay, un cube avec les faces inversées. **Boing.mp3** : un son pour le rebond de la balle avec la palette. **Whoosh.mp3 :** un son quand la balle est désintégrée. Également, **Rebond_grid_map.tga** : une grille 2D pour la caméra. Finalement, une image **splash.png** et **icon.png** pour la version que vous allez exporter à la fin.

10. Sélectionnez les deux ***textures*** (**Rebonds_worldLimit *et* Rebond_grid_map**) dans *Project*.



10.1. Dans l'*Inspector,* changez les paramètres suivants :



10.1.1. ***Type*** : Texture.
10.1.2. ***Alpha from Grayscale*** : Non
10.1.3. ***Alpha Is Tranparency*** : Oui
10.1.4. ***Wrap Mode*** : Repeat
10.1.5. ***Filter Mode*** : Bilinear
10.1.6. ***Aniso Level*** **:** 7
10.1.7. ***Max size*** : 2048
10.1.8. ***Format*** : Truecolor

10.2.Appuyez sur le bouton : « Apply »

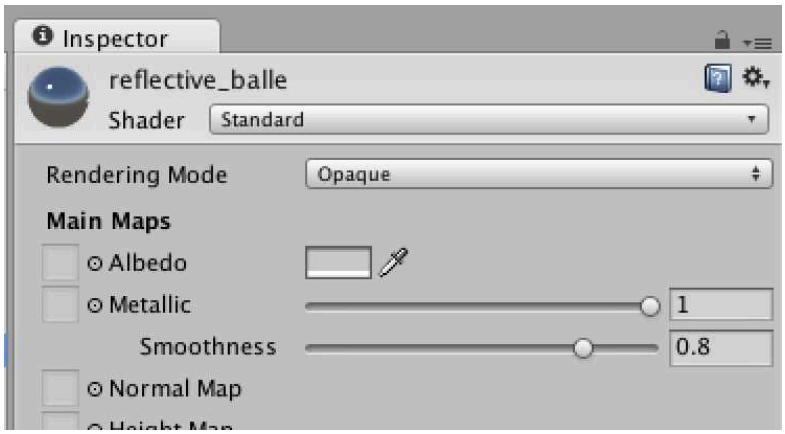
11. Pour la balle, il faut créer un nouveau ***Material***.

11.1. Dans le panneau : *Project,* faites un clic secondaire et sélectionnez :

➢ **Create > Material**

11.2. Nommez-le : **Reflective_balle**

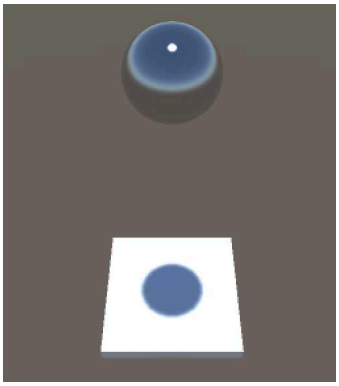
11.3. Dans *l'Inspector*, changez :



11.3.1. ***Shader*** : Standard
11.3.2. ***Rendering Mode*** : Opaque
11.3.3. ***Albedo*** : couleur grise
11.3.4. ***Metallic*** : 1
11.3.5. ***Smoothness*** : 0.8

Pour le moment, laissez le reste tel quel.

En détails...



Le **Shader** : ***Standard*** est une nouveauté d'Unity 5, il est hautement configurable, ce qui lui permet d'être utilisé pour à peu près 95% de vos matériaux.

12. Vous appliquez le ***Material*** à la balle en le glissant sur **BalleRebonds**, plus précisément dans la fenêtre *Scene* ou le panneau *Hierarchy*.

13. Maintenant, pour la limite du monde, glissez l'objet : **Rebonds_worldlimit** dans le panneau *Hierarchy*.

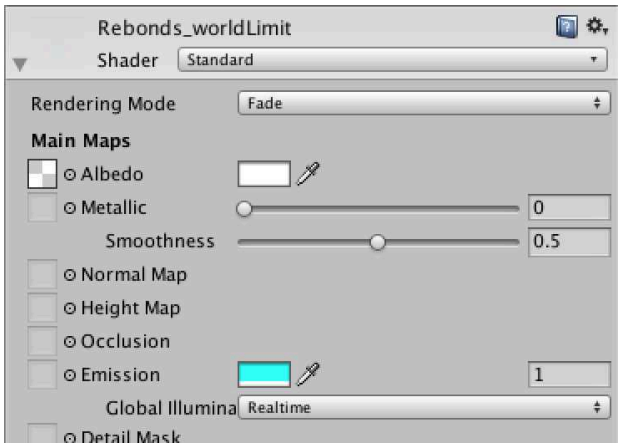
Transform
13.1.1. ***Position*** : **x**=0, **y**=-4, **z**=0
13.1.2. ***Rotation*** : **x**=-90, **y**= 0, **z**=0
13.1.3. ***Scale*** : **x**=1, **y**=1, **z**=1

14. Glissez la texture : **rebonds_worldlimit** sur l'objet.

Grâce à cette manipulation, vous venez de créer un nouveau ***Material***, c'est une deuxième façon de faire!

15. Dans le panneau Project, ouvrez le répertoire **Materials** et sélectionnez : **Rebonds_worldLimit.**

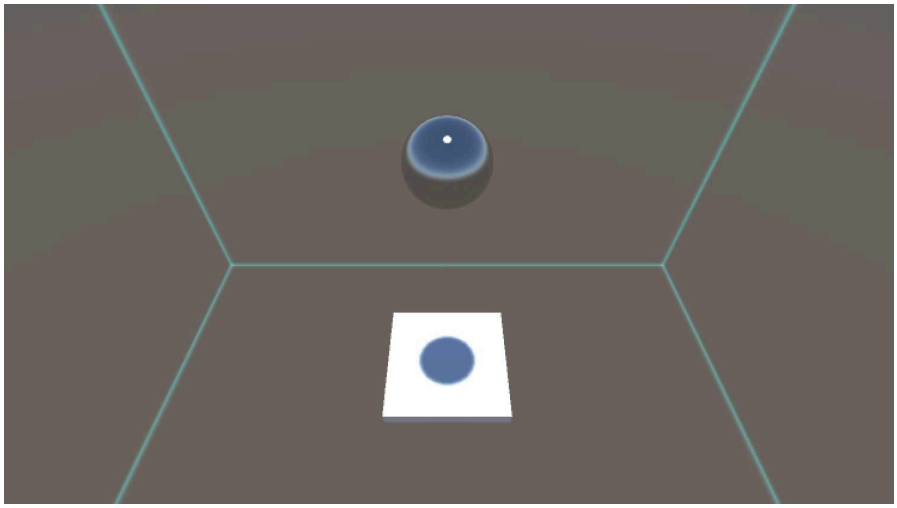
15.1. Dans *l'Inspector,* changez :



15.1.1. ***Shader*** : Standard
15.1.2. ***Rendering Mode*** : Fade
15.1.3. ***Albedo*** : (texture) + **couleur** : Blanche
15.1.4. ***Metallic*** : 0
15.1.5. ***Smoothness*** : 0.5
15.1.6. ***Emission*** : **valeur** : 1 puis comme **couleur** : Bleu pâle

Laissez le reste tel quel.

16. La scène est maintenant transparente.



Sauvegardez votre scène (CTRL+S) ou **(⌘+S)** sur Mac.

Testez le jeu.

Vous constaterez que la balle ne bouge pas. Celà nous indique qu'elle n'a pas de ***Rigidbody***.

Retournez en mode édition.

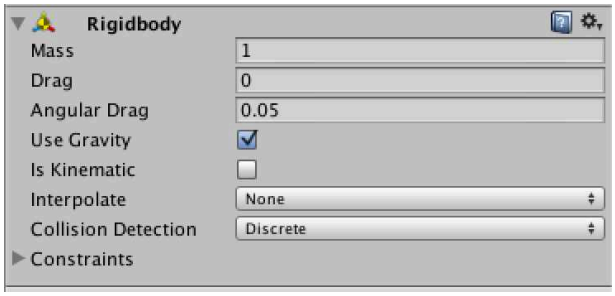
17. Sélectionnez **BalleRebonds** dans la *Hierarchy*.

17.1. À partir de la barre de menus, allez dans :

Component | Physics | Rigidbody

Alternativement, vous pouvez également vous servir du bouton « Add Component » qui se trouve dans *l'Inspector,* puis :

Physics | Rigdbody

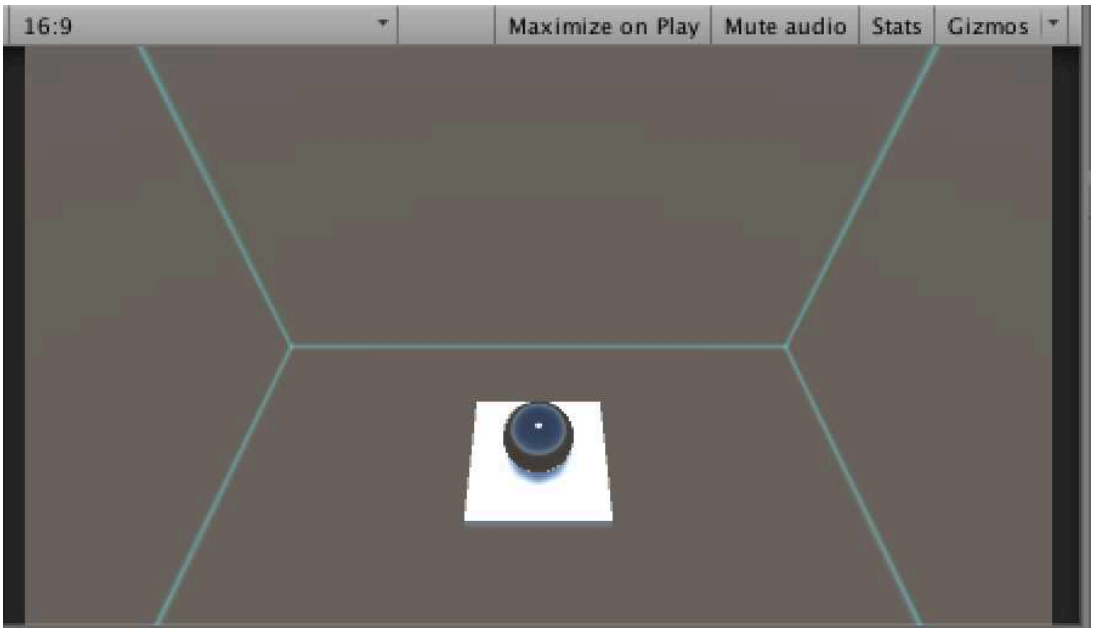


Par défaut, le **RigidBody** utilise la gravité : ***Use Gravity*** sur tous les objets. Vous pouvez désactiver celle-ci et même spécifier que l'animation régit le ***RigidBody,*** au lieu de la physique, avec l'option : ***Is Kinematic.***

Sauvegardez votre scène (CTRL+S) ou **(⌘+S)** sur Mac.

Testez le jeu.

On voit que la balle réagit maintenant avec la gravité. Si celle-ci ne rebondit pas, c'est parce qu'on ne lui a pas assigné de ***Physic Material***.



Retournez en mode édition.

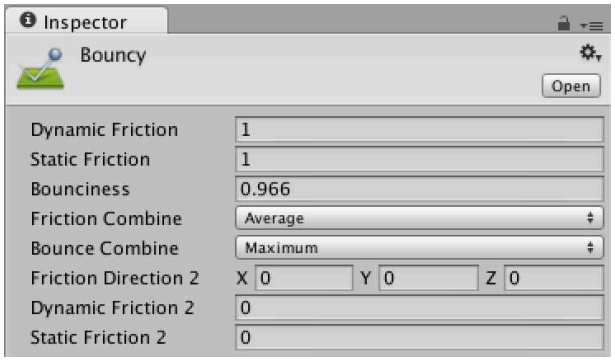
Le ***Physic Material*** permet de définir le comportement de la physique, dans votre univers. Comme les jeux vidéo ne sont pas fidèles à la réalité, vous pouvez ajuster les paramètres à votre guise.

18. Dans le panneau *Project,* faites un clic secondaire puis :

➢ **.Create > Physic Materials**

19. Nommez-le : ***Bouncy***

19.1. Dans *l'Inspector*, changez les propriétés :



Physic Material
19.1.1. ***Dynamic Friction*** **:** 1
19.1.2. ***Static Friction*** **:** 1
19.1.3. ***Bounciness*** **:** 0.966 (La précision est importante).
19.1.4. ***Friction Combine*** **:** Average
19.1.5. ***Bounce Combine*** **:** Maximum

Laissez les autres valeurs à 0.

20. Glissez **Bouncy** sur votre objet **BalleRebond,** dans votre *Hierarchy*.

Sauvegardez votre scène (CTRL+S) ou **(⌘+S)** sur Mac.

Testez le jeu.

La balle rebondit en ayant une propriété spéciale, celle d'ajouter de ne pas perdre de vélocité après un bond.

En tout temps, vous pourrez modifier la propriété du ***Physic Material*** afin de varier le comportement de votre objet.

Retournez en mode édition.

21. Lors du déroulement du jeu, si l'on change un paramètre, comme la Position de la palette, on constate que la balle tombe à l'infini.

Notez-bien!

N'oubliez pas qu'en arrêtant le ***gameplay****, la palette revient à sa position d'origine et tous les changements effectués en mode* ***PLAY ne seront pas sauvegardés !!!***

.

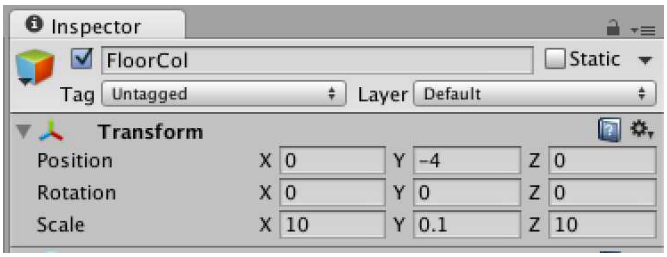
L'environnement

22. Pour les murs, le sol et le plafond, vous allez maintenant ajouter des collisions qui seront ajustées aux dimensions de la scène. Comme le jeu se passe à l'intérieur d'un cube, l'utilisation d'un ***Box Collider***, déclencherait un impact immédiat et le ***Mesh Collider*** ne donnerait pas de meilleurs résultats. Il est plus coûteux que 6 cubes séparés, attachés à la scène.

22.1. Dans la barre de menus, cliquez sur :

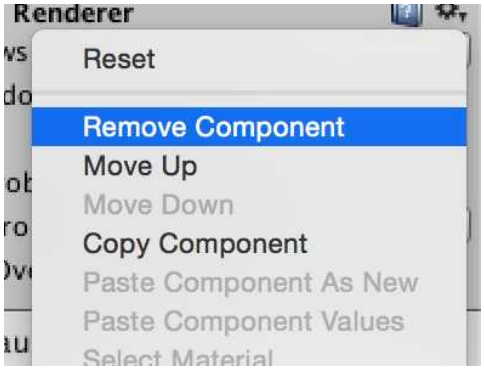
GameObject | 3D Object | Cube

22.2. Dans l'onglet « Inspector », changez les propriétés :

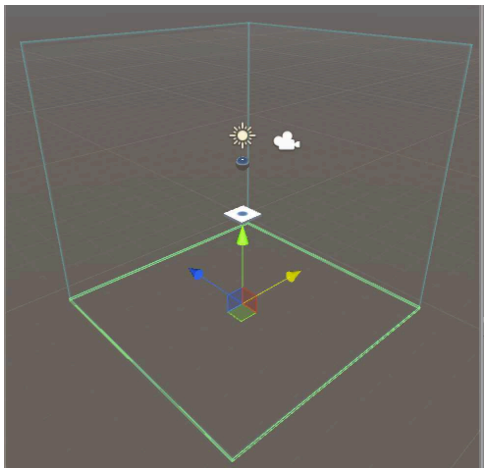


Transform
22.2.1. ***Nom*** : FloorCol
22.2.2. ***Position*** : **x**=0, **y**=-4, **z**=0
22.2.3. ***Rotation*** : **x**=0, **y**= 0, **z**=0
22.2.4. ***Scale*** : **x**=10, **y**=0.1, **z**=10

22.2.5. Effacez le ***Component*** : ***Mesh Renderer*** en cliquant sur l'icône de l'engrenage dans le coin en haut, à droite du ***Component,*** puis en appuyant sur « Remove Component ».



22.2.6. Attachez **FloorCol** à **worldLimit** dans le panneau *Hierarchy* pour le mettre enfant de celui-ci.



Sauvegardez votre scène (CTRL+S) ou **(⌘+S)** sur Mac.

Testez le jeu.

Testez la collision de la balle avec le sol en déplaçant la palette à même la fenêtre *Scene,* hors de la trajectoire de la balle.

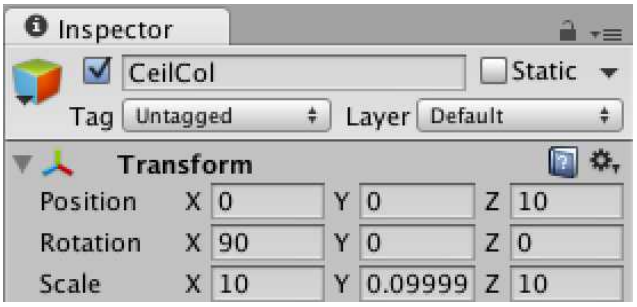
La balle devrait rebondir sur le sol.

Retournez en mode édition.

23. Dupliquez **FloorCol** en le sélectionnant et en appuyant sur (**CTRL+D**) ou (**⌘+D**) sur Mac.

23.1. Renommez cet objet : **CeilCol** (Plafond).

23.2. Dans *l'Inspector,* changez :

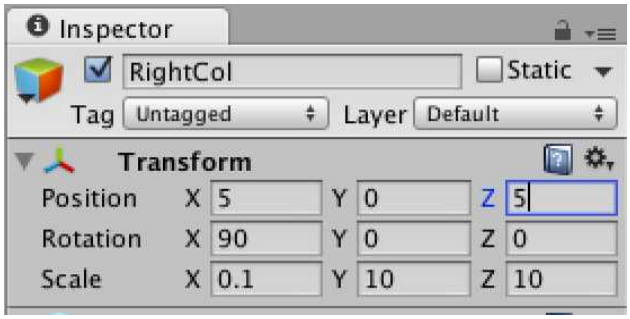


Transform
23.2.1. ***Position*** **: X=**0, **Y**=0, **Z**=10
(relativement à son parent **worldLimit**)

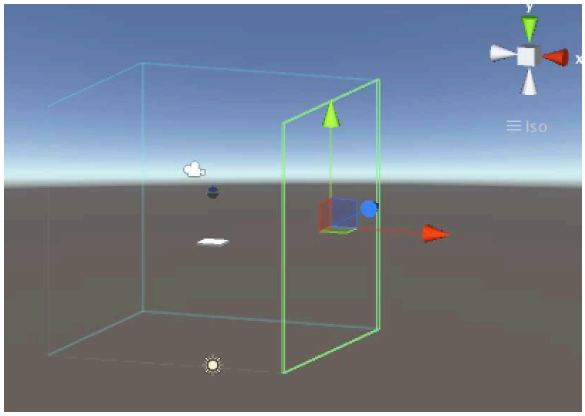
24. Dupliquez **CeilCol** en le sélectionnant et en appuyant sur (**CTRL+D**) ou (**⌘+D**) sur Mac.

24.1. Renommez cet objet : **RightCol** (Droit).

24.2. Dans *l'Inspector,* changez :



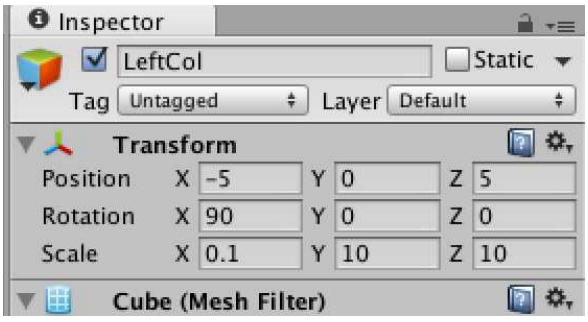
Transform
24.2.1. ***Position*** **: X=**5, **Y**=0, **Z**=5 (relativement à son parent **worldLimit**)
24.2.2. ***Scale*** **: X=**0.1, **Y=**10, **Z=**10



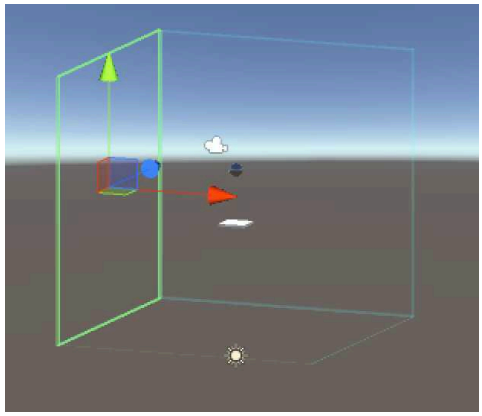
25. Dupliquez **RightCol** en le sélectionnant et en appuyant sur (**CTRL+D**) ou (**⌘+D**) sur Mac.

25.1. Renommez cet objet : **LeftCol** (Gauche).

25.2. Dans *l'Inspector,* changez :



Transform
25.2.1. ***Position*** **: X= -**5, **Y**= 0, **Z**= 5 (relativement à son parent **worldLimit**)
25.2.2. ***Scale*** **: X=** 0.1, **Y=** 10, **Z=** 10

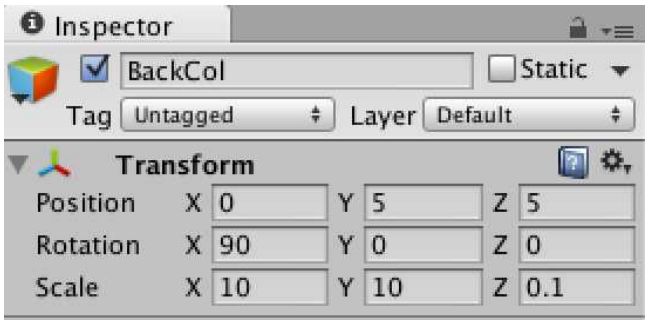


Il ne reste plus que deux murs.

26. Dupliquez **LeftCol** en le sélectionnant et en appuyant sur (**CTRL+D**) ou (**⌘+D**) sur Mac.

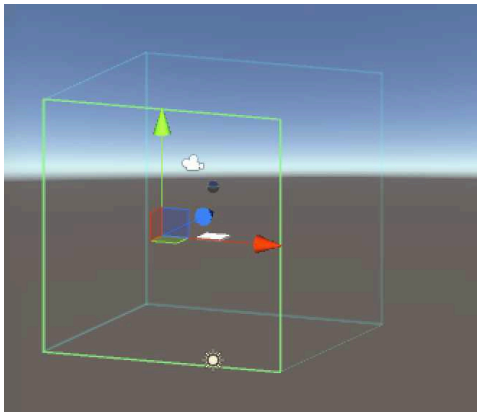
26.1. Renommez cet objet : **BackCol** (Arrière).

26.2. Dans *l'Inspector,* changez :



Transform
26.2.1. ***Position*** **: X=** 0, **Y=** 5, **Z=** 5 (relativement à son parent **worldLimit**)

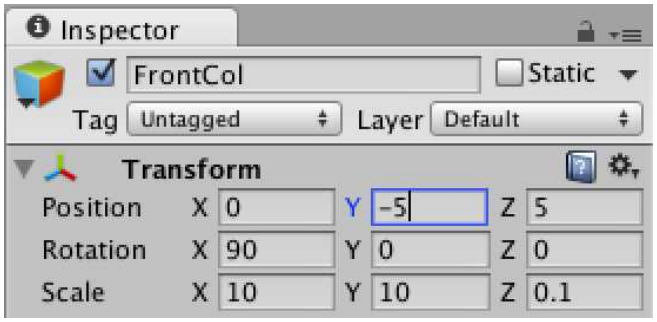
26.2.2. ***Scale*** **: X=** 10, **Y=** 10, **Z=** 0.1



27. Dupliquez **BackCol** en le sélectionnant et en appuyant sur (**CTRL+D**) ou (**⌘+D**) sur Mac.

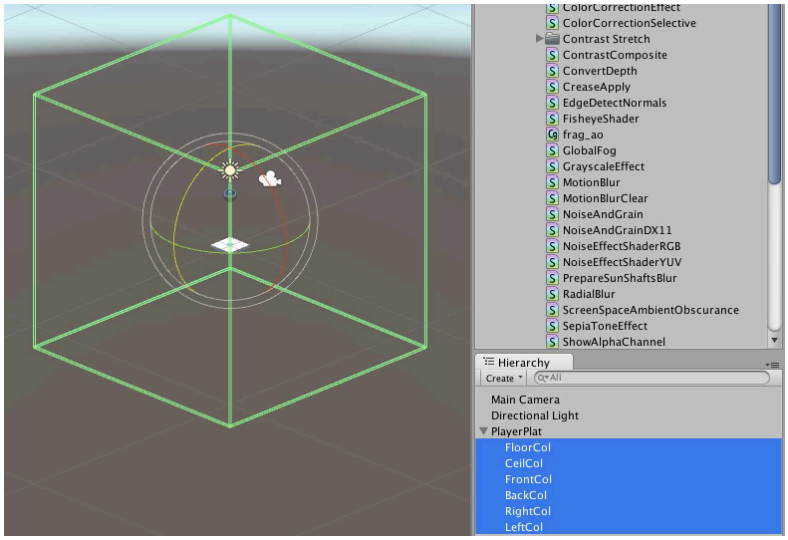
27.1. Renommez cet objet : **FrontCol** (Avant).

27.2. Dans *l'Inspector,* changez :



Transform
27.2.1. ***Position*** **: X=** 0, **Y=** -5, **Z=** 5 (relativement à son parent **worldLimit**)
27.2.2. ***Scale*** **: X=** 10, **Y=** 10, **Z=** 0.1

Si vous sélectionnez les différents murs, vous devriez voir ceci dans votre : *Scene.*



Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Maintenant, vous avez une bonne partie des éléments pour le jeu. Il vous manque des points de repère pour le gameplay, un radar dans le coin de l'écran afin de garder un œil sur votre palette. La balle vue du haut aiderait le gameplay.

Vous allez utiliser l'image de grille 2D qui servira de point de repère. Pour cela, il faut ajouter une surface pour votre grille.

28. Dans la barre de menus, cliquez sur :

GameObject | 3D Object | Plane

28.1. Nommez-le : **GrilleMap**

28.2. Dans *l'Inspector,* changez les paramètres :

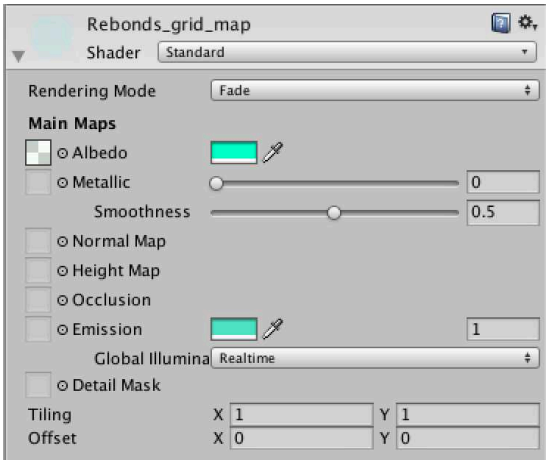
Transform
28.2.1. ***Position*** **: X=** 0, **Y=** 0, **Z=** 0
28.2.2. ***Scale*** : **X=** 1, **Y=** 1, **Z=** 1

29. Glissez la texture **Rebonds_grid_map**, de *Project* sur l'objet: **GrilleMap** dans la *Scene*.

30. Un nouveau ***Material*** sera créé.

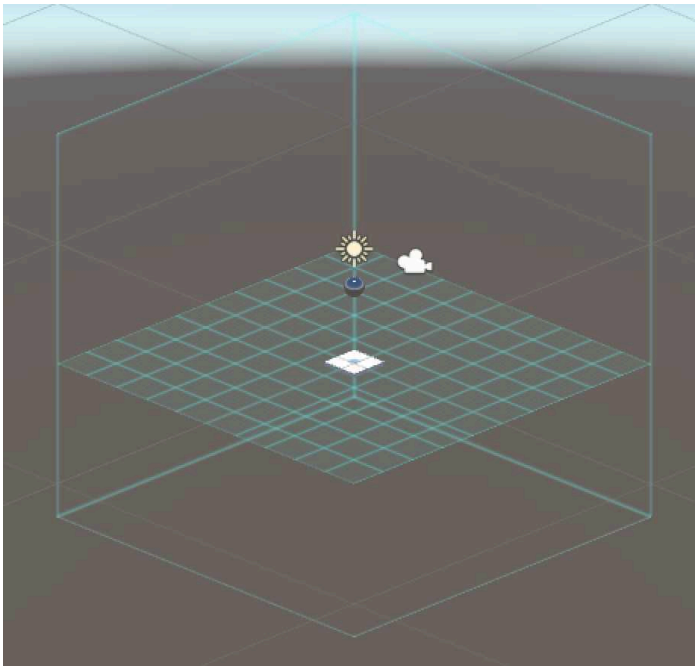
31. Sélectionnez ce ***Material*** dans la section *Project.*

31.1. Modifiez les paramètres suivants dans *l'Inspector* :



31.1.1.***Shader*** : Standard
31.1.2.***Rendering Mode*** : Fade
31.1.3.***Albedo*** : (Texture), **couleur** : Bleu pâle
31.1.4.***Metallic*** : 0
31.1.5.***Smoothness*** : 0.5
31.1.6.***Emission*** : **valeur** : 1, **couleur** Bleu pâle

Laissez le reste tel quel.



32. Supprimez le ***Component*** : ***Mesh Collider*** dans *l'Inspector*, en faisant un clic secondaire sur le ***Component*** puis en appuyant sur « Remove Component »

Le radar

Vous allez créer une deuxième caméra (2D) qui va filmer la scène du dessus. Le résultat sera superposé dans la vue 3D afin de créer l'effet escompté.

33. Dans la barre de menus, cliquez sur :

GameObject | Camera

34. Sélectionnez la nouvelle caméra.

34.1. Nommez-la : **Top Camera.**

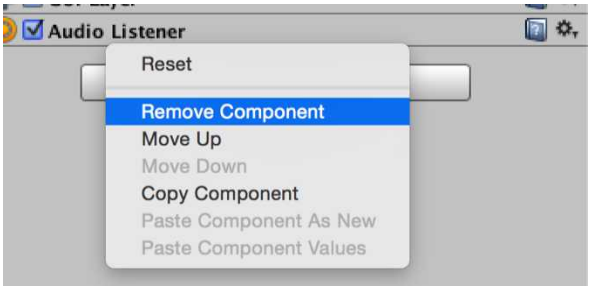
34.2. Dans *l'Inspector*, changez les paramètres suivants :

Transform
34.2.1. ***Position*** : **X**=0, **Y**=7.75, **Z**=0
34.2.2. ***Rotation*** : **X**=90, **Y**=0, **Z**=0

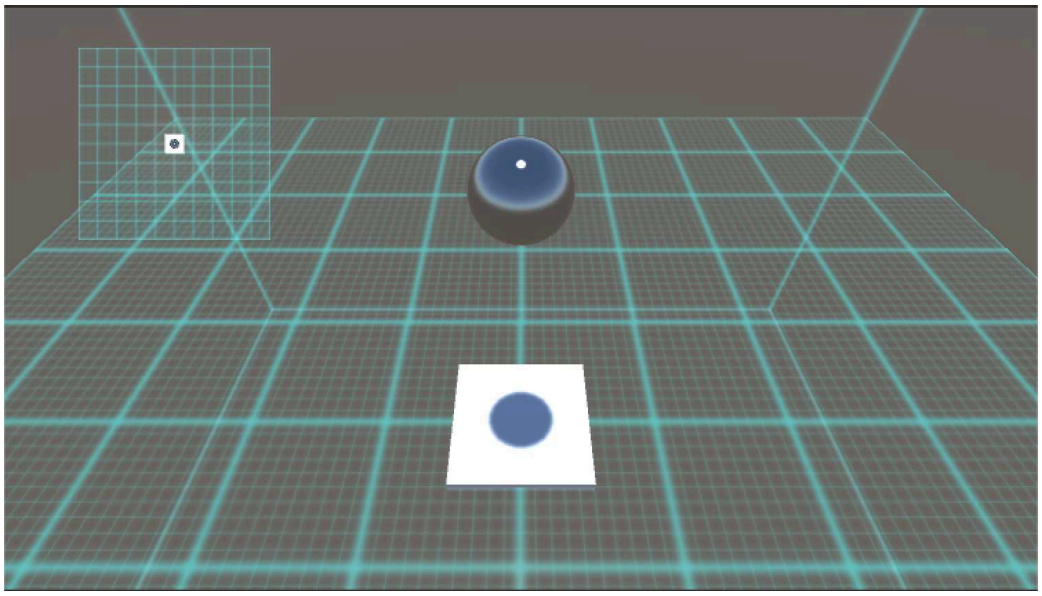
Camera
34.2.3. ***Clear Flags*** **:** Depth only
34.2.4. ***Culling Mask*** **:** Everything
34.2.5. ***Projection*** : Orthographic (pour le 2D)
34.2.6. ***Size*** : 5

34.2.7. ***Clipping*** : Near = -100, Far = 100
34.2.8. ***Normalized View Port Rectangle*** : X = 0, W=0.33, Y=0.6, H=0.33
34.2.9. ***Depth*** : 1

35. Retirez *l'**Audio Listener*** de **Top Camera** avec un clic secondaire sur ce ***Component*** puis appuyez sur « Remove Component ».

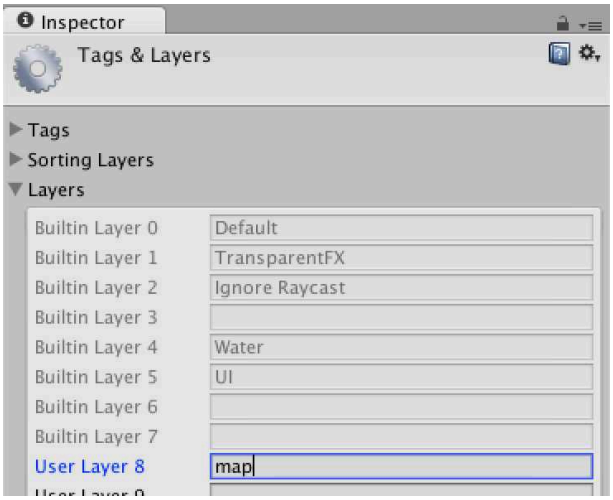


Pour notre **Main Camera**, il est important de ne pas voir la grille dans la vue 3D.



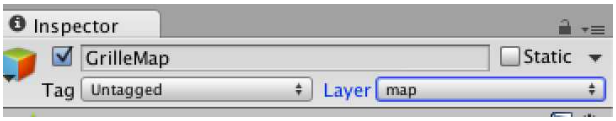
36. Dans la barre de menus, allez dans :

Edit | Project Settings | Tags & Layers



36.1. Ajoutez le texte **map** dans le champ ***Layers*** / **User Layer 8.**

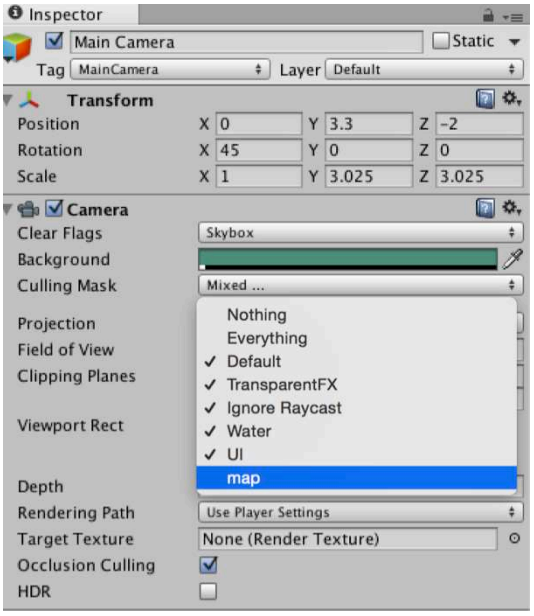
37. Dans votre *Hierarchy*, sélectionnez : **GrilleMap.**



37.1. Dans *l'Inspector*, en-dessous du nom, changez le ***Layer*** pour : **map**.

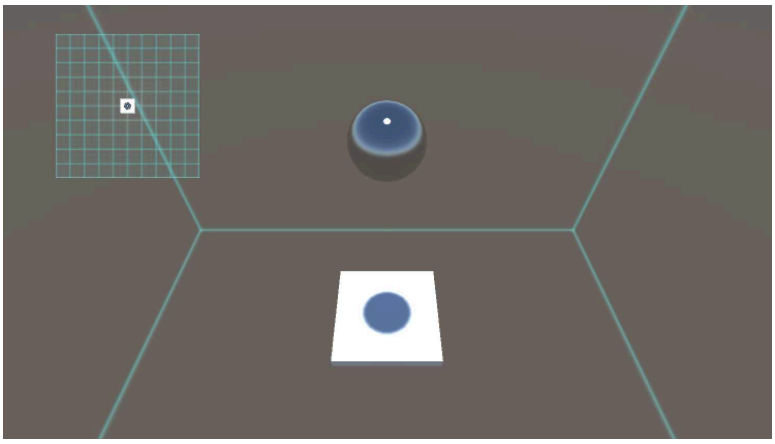
Vous allez maintenant indiquer à la caméra principale de ne pas faire le rendu des objets sur ce ***Layer***.

38. Sélectionnez **Main Camera.**



38.1. Dans *l'Inspector*, changez :

38.1.1. ***Culling mask*** : Retirez **map** de la liste.



Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac!

Vos préparatifs sont presque complets. Vous allez ajouter un : ***UI Text*** pour conserver une trace du pointage.

Pour ajouter un ***UI Text*** dans la scène :

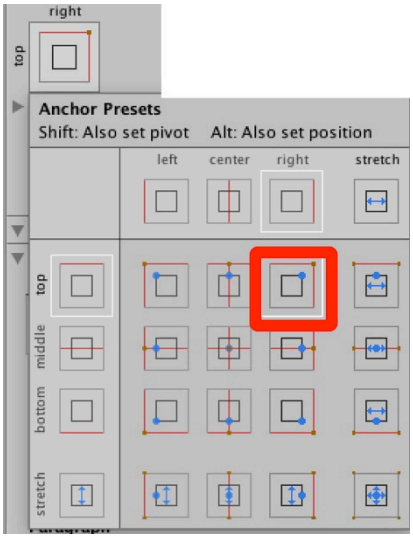
39. Dans la barre de menus, cliquez sur :

GameObject | UI | Text

39.1. Nommez-le: **GUI_Score.**

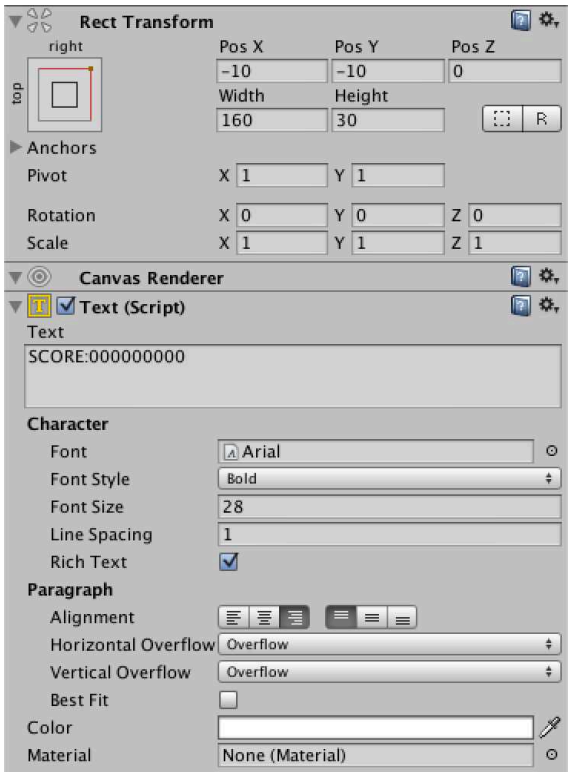
39.2. Dans *l'Inspector,* changez :

Rect Transform
39.2.1. ***Anchor Presets*** : ancrage **Top | Right**



39.2.2. ***Pos* X** = -10**, *Pos* Y** = -10**, *Pos* Z** = 0
(Il est préférable de garder ce paramètre pour la fin).

39.2.3. ***Pivot* : X** = 1**, Y** = 1



Text (Script)
***Character* :**
39.2.4. ***Text*** : SCORE : 000000000
39.2.5. ***Font*** : Arial
39.2.6. ***Font Style*** : Bold
39.2.7. ***Font Size*** : 28
39.2.8. ***Line Spacing*** : 1
39.2.9. ***Righ Text*** : Oui

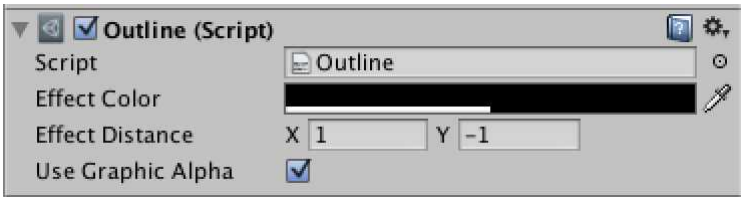
***Paragraph* :**
39.2.10. ***Alignment*** : Droite, Haut

39.2.11. ***Horizontal Overflow***: Overflow
39.2.12. ***Vertical Overflow*** : Overflow
39.2.13. ***Best Fit*** : non
39.2.14. ***Color*** : Blanc
39.2.15. ***Material*** : Aucun

Afin d'augmenter la lisibilité, vous allez ajouter un effet de contour autour du texte.

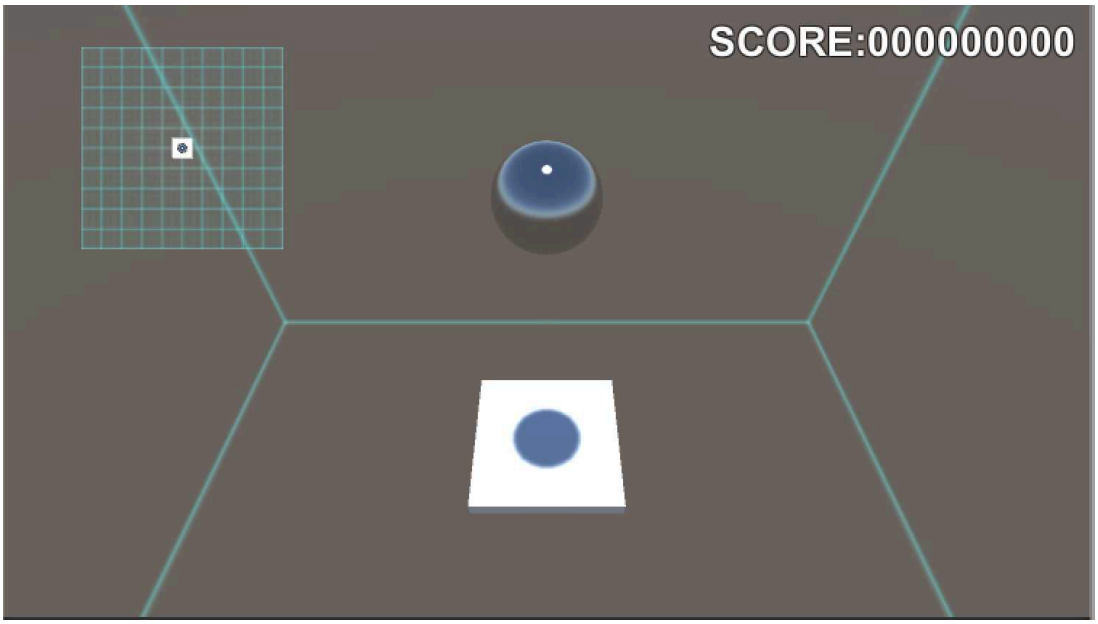
39.3. Dans *l'Inspector,* appuyez sur le bouton « Add Component » puis :

UI | Effects | Outline



Vous pouvez changer les polices et leurs propriétés, mais pour faciliter l'exercice, vous allez conserver la police : **Arial**.

Voici le résultat que vous devriez obtenir :



Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Comme les éléments sont en place, vous allez ajouter quelques scripts pour votre **jouabilité,** mais avant de se lancer dans la programmation, vous allez parcourir les bases du JavaScript.

Si vous avez de l'expérience en programmation, vous pourriez aller directement au **chapitre 4 – Procédure : Programmation de Rebonds** afin de terminer cet exercice.

Chapitre 3 - Premiers pas en programmation JavaScript

Matière couverte dans ce chapitre

- **Le pseudo-code**
- **La syntaxe**
- **Les variables**
- **Les opérations**
- **Les conditions**
- **Les boucles**
- **Les fonctions**
- **La surcharge de fonctions (sur définition)**
- **Les fonctions et les variables intégrées**
- **Les classes**
- **L'héritage**

Dans ce chapitre, vous allez découvrir l'introduction à la programmation JavaScript. Aucun prérequis n'est nécessaire pour cette section.

Le pseudo-code

Qu'est-ce que le pseudo-code ?

Le pseudo-code est de la programmation en langage humain. Avant d'amorcer cette notion, il est cependant important d'avoir une bonne préparation et évaluation de nos besoins avant d'écrire quoi que ce soit.

Le planning est plus important que l'exécution !

Écrire un pseudo-code, c'est exactement comme écrire une recette de cuisine, on commence par une liste d'ingrédients (les variables) et ensuite vient l'écriture de la préparation (les instructions) dans un langage qui peut être compris par le commun des mortels.

Le test du corridor

La meilleure méthode pour connaître si le pseudo-code est vulgarisé de façon appropriée, c'est de faire, ce que l'on appelle, le test du corridor. En d'autres mots, est-ce que la lecture du pseudo-code peut s'effectuer par quelqu'un qui n'a aucune expérience en programmation ? Est-ce qu'en suivant les étapes, ligne par ligne, le lecteur peut comprendre la logique du programme ?

L'exemple du retrait au distributeur de billets / guichet automatique bancaire

Voici un exemple simple d'un pseudo-code pour expliquer le déroulement d'un retrait au guichet automatique bancaire :

Variables :

- **Numéro de la carte :** Nombre de 16 chiffres sur la bande magnétique de la carte
- **NIP :** Code numérique de 4 à 6 chiffres
- **Nombre d'essais pour le NIP :** Nombre
- **Solde du compte :** Nombre
- **Montant à retirer :** Nombre
- **Imprimer reçu :** (vrai ou faux)

Instructions :

1. *Attendre l'insertion d'une carte.*
2. *Lire la carte.*
3. *Demander le **NIP** à l'usager.*
4. *Attendre le **NIP.***

5. *Si le NIP **est égal** au **NIP de la carte,** passe à la partie retrait **sinon** :*
6. *On augmente **le nombre d'essais pour le NIP** de +1.*
7. *Si le nombre d'essais **est égal à 3,** alors garde la carte sinon :*
8. *Aller à **l'étape 3**.*

Fonction *Retrait* :

1. *Afficher le **solde du compte.***
2. *Demander le **montant** à retirer.*
3. *Attendre le **montant.***
4. ***Solde du compte** = **Solde** – **montant** à retirer.*
5. *Donner **l'argent** au client.*
6. *Afficher le nouveau **solde** du compte.*
7. *Demander s'il veut imprimer un **reçu.***
8. *Attendre **réponse.***
9. *Si oui : imprimer **reçu.***
10. *Éjecter la carte.*

L'exemple précédent vulgarise un distributeur de billets qui ne fait qu'une seule action, les retraits, mais il est facile d'extrapoler l'exercice et d'ajouter d'autres modules, comme les dépôts, ou consulter le solde. Chaque partie représente une ***fonction*** qui sert à exécuter une série de commandes afin d'atteindre un objectif. Par exemple, faire un retrait au guichet automatique bancaire.

Pourquoi faire un pseudo-code ?

Après plusieurs années d'expérience en programmation, il est facile d'omettre le pseudo- code, car la programmation peut devenir instinctive. Par contre, être bien préparé peut permettre de réaliser en 100 lignes de codes ce que l'on aurait autrement effectué en 1000 lignes.

En utilisant cette démarche, on risque moins de commettre des bogues, de se perdre dans sa logique et le travail d'équipe est plus efficace. Dans les grandes entreprises et sur les projets non triviaux, le planning est primordial. Chaque équipe doit coder une partie vitale du projet pour ensuite collaborer avec d'autres modules créés par d'autres gens qui n'ont jamais vu au préalable l'autre partie du travail ou le projet, dans son intégralité.

Parfois, le pseudo-code peut nous faire réaliser l'ampleur d'un travail avant son exécution, car en être conscient permet de prendre les bonnes décisions, afin de maximiser le temps de production. Ceci nous permet également de connaître où couper et quels modules devraient être achetés plutôt que programmés à l'interne.

La syntaxe

Les bases

Ces bases sont valides pour les langages dérivés du **C** comme le **C++, C#, JavaScript et Actionscript.**

Le Basic, Fortran ou le Delphi par exemple, utilisent la même logique mais avec une syntaxe différente.

1. Toutes les commandes doivent se terminer par un point-virgule (**;**)

2. On regroupe une série d'instructions en les encadrant avec des accolades (**{ }**)

3. On déclare une nouvelle variable avec l'instruction (***var***). Une fois déclarée, on ne doit plus réutiliser cette instruction pour cette même variable et on accède à ses propriétés en utilisant son nom.

4. On regroupe les différents blocs que l'on veut utiliser plusieurs fois dans le code avec des **fonctions** : (***function***)

function **nomFonction (*variables*) { *instructions* }**

Ces fonctions peuvent recevoir et passer des variables en paramètres afin de les rendre réutilisables dans différents contextes.

5. Certaines variables contiennent plusieurs propriétés. On accède aux propriétés d'un objet en utilisant le (**.**) entre le nom de la variable et la propriété que l'on veut accéder. **Exemple** : **monObjet**.***height***;

Me retourne la hauteur de **monObjet**.

6. On peut mettre des commentaires dans le code afin d'améliorer sa compréhension et parTager des informations utiles entre les programmeurs. Pour mettre un texte en commentaires, on place (//) devant le texte en question. Si vous avez plusieurs lignes à mettre en commentaires, il suffit de les encadrer avec (/*) et (*/)

7. Plusieurs langages de programmation dont le JavaScript est sensible aux majuscules, donc une variable nommée : **LaVariable**, **Lavariable** ou **laVariable** par exemple, sont toutes les trois considérées comme des ***variables différentes***.

Les variables

Les variables sont des contenants qui vont renfermer les différents paramètres du jeu.

Vous pouvez utiliser l'appellation que vous désirez pour vos variables en autant qu'ils ne contiennent aucun espace, que le premier caractère n'est pas un chiffre, que vous utilisez uniquement des lettres, des chiffres ou un trait de soulignement (_) et que le nom que vous utilisez n'est pas réservé par le système.

Pour créer une nouvelle variable, utilisez le mot clé : (**var**).

Nommez votre première variable : **fruit.**

```
var fruit:String;
```

Voilà votre première variable finie ! Vous devez spécifier quel type de variable vous voulez utiliser soit en le déclarant explicitement comme dans le cas précédent ou en plaçant quelque chose dans celle-ci.

Dans l'exemple précédent, vous avez spécifié le type : ***String***. Les types les plus communs sont : ***String*** (texte), ***int*** (nombre entier), ***float*** (nombre réel), ***boolean*** (booléen) et ***Array*** (tableau). Notez que la capitalisation de ces types de variables est importante et définie par des conventions.

Si vous vous demandez pourquoi il y a un point-virgule (**;**) à la fin de la ligne, c'est parce qu'en JavaScript on doit utiliser le point-virgule pour indiquer au système où l'instruction se termine.

Dans l'exemple précédent vous avez créé une variable vide, on ajoute une valeur à une variable comme ceci :

```
fruit = "pomme";
```

Maintenant votre fruit contient un texte : **pomme**

Notez-bien!

*Vous pouvez déclarer une **variable** et lui assigner une valeur en même temps.*

.

```
var fruit = "pomme";
```

Par contre, une fois la variable déclarée, on ne doit plus la déclarer à nouveau mais vous pouvez changer son contenu (en autant que celui-ci est du même type que l'original).

Lorsque vous déclarez une variable et que vous lui assignez une valeur en même temps, vous n'avez pas à spécifier explicitement le type de ***variable***. Le compilateur voit une valeur de type ***String***, alors, il assume que la variable doit également être de type ***String***. Vous pouvez quand même spécifier le type de variable, si vous le désirez. Dans bien des cas, le code devient plus clair.

```
var fruit:String = "pomme";
```

Dans les deux derniers exemples, Unity interprète les deux variables de façon identique. Une variable de type ***String*** contient le texte : **pomme**. Explicitement ou non la variable **fruit** sera toujours considérée comme un ***String*** et ne pourra pas changer de type en cours d'exécution.

Notez-bien!

*On doit toujours encadrer notre valeur texte entre guillemets (") ou apostrophes ('). Si vous pensez utiliser un de ces symboles comme valeur, utilisez l'autre pour encadrer votre texte : "**l'autre jour**" ou '**10**'*

.

```
fruit = "banane"; // Ça fonctionne correctement
fruit = 1; // Ici il y a une erreur, car 1 n'est pas un String.
```

En plus des textes (***String***), les ***variables*** peuvent contenir des nombres :

```
var nombre:int = 100;
```

Ce nombre est entier (***int*** pour **integer**), ce qui veut dire qu'on ne peut pas mettre de décimale dans ce nombre.

Il existe également un type pour les nombres réels (***float***) qui peut contenir des décimales.

Notez-bien!

*Contrairement aux textes (**String**) les nombres n'utilisent pas de guillemets (") ou d'apostrophes ('), si vous le faites, vous vous retrouverez avec un texte qui contient des caractères numériques.*

```
var nbReel:float = 10.5;
```

Une variable peut également contenir une valeur booléenne (***boolean***). Ce type contient uniquement : vrai/faux (***true/false***).

```
var gameOver:boolean = false;
```

La syntaxe pour un tableau est d'ajouter des crochets (**[]**) à la fin de la déclaration explicite d'un type de variable. Alors, si vous voulez plusieurs textes (***String***) dans un tableau, vous déclarez ***String[].***

Notez-bien!

*Habituellement, une variable ne contient qu'un seul élément à la fois. Mais vous pouvez également créer une variable qui contient plusieurs éléments dans un tableau de données (**Array**).*

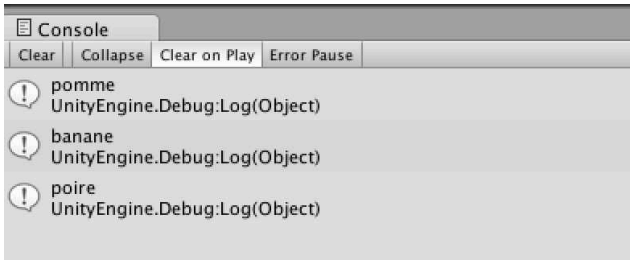
```
var fruits:String[] = ["pomme","banane","poire"];
```

Cette variable contient trois fruits.

Si vous voulez voir le contenu de **fruits,** vous pouvez l'afficher dans la console de débogage. Par exemple :

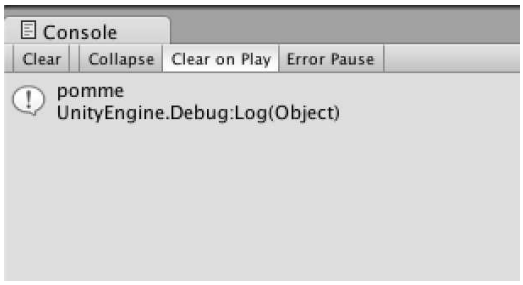
```
for (var cpt=0; cpt<fruits.length; cpt++){
 Debug.Log(fruits[cpt]);
}
```

Ce code devrait afficher :



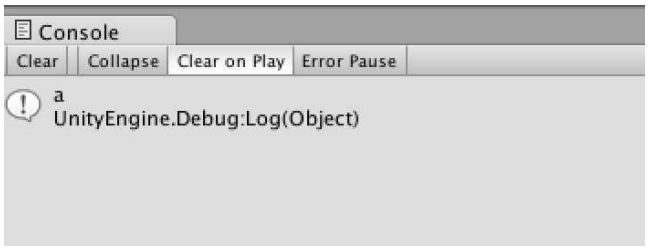
Pour récupérer un élément dans un tableau, on utilise son numéro d'index. Les index débutent à **0**. Alors l'index **0** est le premier élément dans votre tableau.

```
var fruits:String[] = ["pomme","banane","poire"];
var fruit1 = fruits[0];
Debug.Log(fruit1);
```



Vous pouvez également utiliser les index avec les textes (***String***) afin de récupérer une lettre à la fois. Le code suivant affiche la première lettre du mot **allo**, la lettre : **a.**

```
var monTexte:String = "allo";
Debug.Log(monTexte[0]);
```

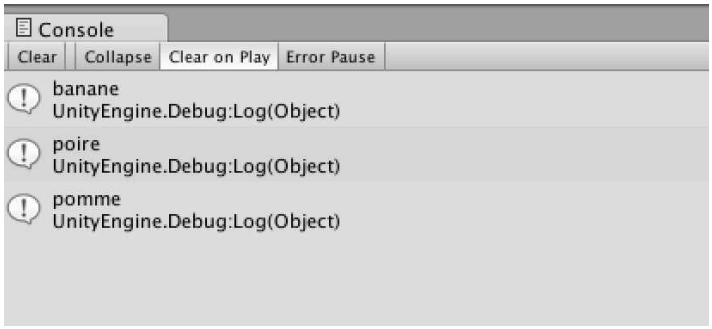


Notez-bien!

*Il existe un autre type de tableau, le type : **Array.** Son avantage, c'est qu'il est dynamique et l'on peut y ajouter, enlever, réorganiser et archiver différents types d'objets. Par contre, on ne peut pas voir le contenu de ce type de tableau dans l'Inspector, par conséquent, on ne peut y ajouter des éléments de la scène.*

Dans le prochain exemple, il faut réorganiser les différents éléments du tableau en ordre alphabétique avant d'en afficher son contenu.

```
var monTableau:Array = new Array("pomme", "banane", "poire");
monTableau.Sort();

for (var cpt=0; cpt<monTableau.length;cpt++){
 Debug.Log(monTableau[cpt]);
}
```



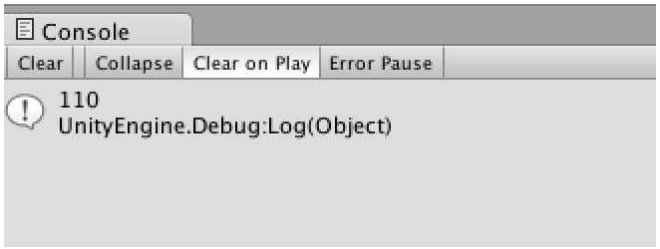
Comme vous pouvez le constater, les trois textes ne sont pas dans l'ordre original.

Les opérations

Maintenant, vous allez voir un autre usage des variables.

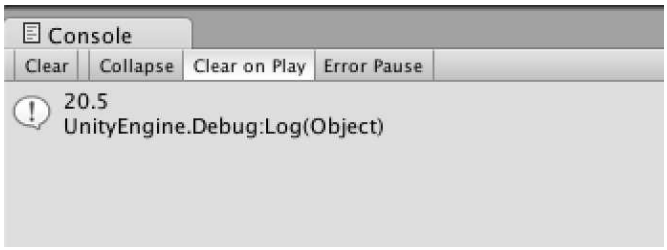
On peut par exemple, additionner les variables :

```
var nombre1 = 10;
var nombre2 = 100;
var total = nombre1 + nombre2;
Debug.Log(total);
```



Vous pouvez additionner un nombre entier (***int***) et un nombre réel (***float***), le résultat devient un nombre réel (***float***).

```
var nombreA = 10;
var nombreB = 10.5;
var nouvTotal = nombreA + nombreB;
Debug.Log(nouvTotal);
```

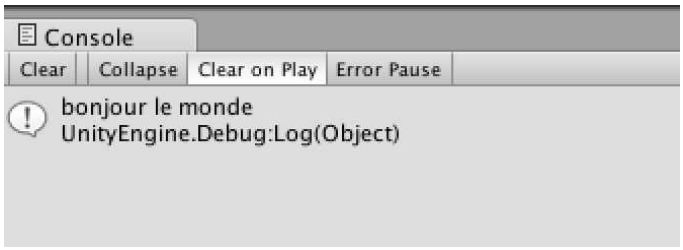


Vous pouvez également soustraire (**-**), multiplier (*****) ou diviser (**/**) des nombres entiers et réels.

Il y a également l'opérateur modulo (**%**) qui nous donne le reste d'une division.

Vous pouvez également additionner deux textes ensemble afin de les regrouper dans une variable.

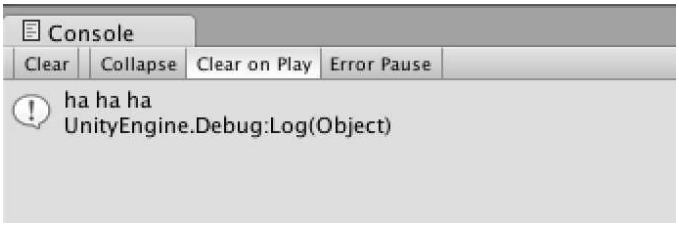
```
var texte1 = "bonjour ";
var texte2 = "le monde";
var texte3 = texte1 + texte2;
Debug.Log(texte3);
```



Si vous additionnez un **nombre** et un **texte** ensemble, le résultat deviendra un texte (***String***).

Vous pouvez également multiplier des textes.

```
var rire = "ha " * 3;
Debug.Log(rire);
```



Malheureusement, vous ne pouvez pas soustraire ou diviser des textes. Il existe d'autres moyens pour séparer ceux-ci, mais c'est un peu plus ardu.

Vous allez maintenant voir comment incrémenter la valeur d'un nombre (augmenter sa valeur de +1).

Tout d'abord, vous allez déclarer une variable et la mettre à 1 :

```
var nombre = 1;
```

Il existe maintenant plusieurs façons d'incrémenter cette variable.

```
nombre = nombre + 1;
```

La commande précédente ajoute 1 à « nombre » celui-ci devient égal à 2.

Puisque les programmeurs sont souvent expéditifs, ils ont décidé de trouver une méthode plus concise. Les abréviations ont donc été créées.

```
nombre += 1;
```

Cette commande est l'abrégée de *nombre = nombre + 1;*

Il existe également une méthode encore plus concise :

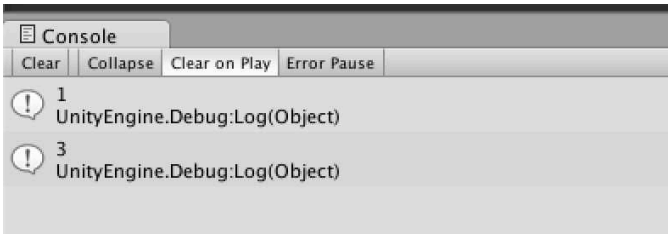
```
nombre++;
```

C'est une façon plus rapide d'augmenter la valeur d'une variable de 1. Vous pouvez également l'écrire comme ceci :

```
++nombre;
```

La différence entre ces deux dernières méthodes, c'est lorsqu'on place (++) après la variable, sa valeur est retournée puis incrémentée. Quand le (++) est devant le nom de la variable, alors celle-ci est d'abord incrémentée puis retournée. À moins de récupérer la valeur de la variable à l'incrémentation, il n'y a pas de différence.

```
var nombre = 1;
Debug.Log (nombre++);
Debug.Log (++nombre);
```



Mais où est le nombre 2 ? Il se trouve entre les deux appels au débogueur. Comme mentionné plus haut, le nombre est d'abord retourné puis incrémenté. La seconde fois, il est incrémenté puis retourné, c'est pourquoi le nombre 2 n'est jamais retourné.

Vous pouvez faire la même démarche avec les soustractions :

```
--nombre;
nombre--;
```

Pour les multiplications et les divisions, vous devez utiliser une méthode plus longue :

```
nombre = nombre/2;
nombre = nombre*2;
nombre /= 2;
nombre *= 2;
```

Les conditions

Il existe deux types de conditions, si (***if***) et (***switch*** & ***case***). Une condition évalue si le résultat de celle-ci est vrai (***true***). Si oui, alors on fait quelque chose…

Vous pouvez évaluer deux valeurs en utilisant les signes (**==**) : *est égal à*…

nombre == 10 évalue si nombre est égal à 10, si c'est (**vrai**), il exécute le bloc de codes attaché à la condition. Si c'est (**faux**), on passe à la prochaine instruction, sans exécuter le code attaché à la condition.

Notez-bien!

Il est important de se rappeler qu'on utilise deux symboles égal (==) pour comparer des variables et des valeurs. On utilise un symbole égal (=) quand on veut assigner une valeur à une variable.

Le code suivant crée une variable et la met à (***true***) vrai, ensuite vous devez valider si la condition est vraie pour pouvoir afficher un texte dans la console de débogage.

```
var partieEnCours = true;
if (partieEnCours == true)
 Debug.Log ("La partie est en cours");
```

Le code précédent est redondant, car la variable **partieEnCours** est déjà booléenne. Il n'y a aucune raison de valider si la variable **est égal à** (**==**) vrai, il faut simplement la vérifier.

```
var partieEnCours = true;
if (partieEnCours)
 Debug.Log ("La partie est en cours");
```

Vous vous demandez probablement pourquoi nous n'avons pas utilisé de point-virgule (**;**) après : **partieEnCours**, c'est parce que techniquement seulement la première moitié de l'instruction se retrouve sur cette ligne. Nous aurions également pu écrire l'instruction comme ceci :

```
if (partieEnCours) Debug.Log ("La partie est en cours");
```

ou

```
if (partieEnCours){
      Debug.Log ("La partie est en cours");
}
```

Vous pouvez également exécuter un bloc de codes après une condition. Pour ce faire, vous devez absolument utiliser les accolades (**{ }**), celles-ci sont optionnelles si vous n'avez qu'une seule instruction.

Notez-bien!

Le point-virgule (;) n'est pas requis après une accolade.

Vous pouvez également raccourcir une condition négative :

```
if (partieEnCours == false)
      Debug.Log ("La partie n'est pas en cours");
```

On peut l'écrire comme ceci:

```
if (!partieEnCours)
      Debug.Log ("La partie n'est pas en cours");
```

Le point d'exclamation (!) veut dire **n'est pas** en JavaScript. En l'utilisant avec un booléen nous pouvons éviter la redondance.

Vous pouvez également utiliser le point d'exclamation avec d'autres types de variables, par exemple un nombre :

```
if (reponseA != "A")
  Debug.Log ("Mauvaise réponse");
```

Vous pouvez aussi valider si l'équation est plus petite ou plus grande qu'une valeur :

```
var temperature = 37.0;
if (temperature > 36.1)
      Debug.Log ("Danger d'hypothermie");
else if (temperature < 37.8)
      Debug.Log ("Danger fièvre");
else
      Debug.Log ("Normal");
```

Avez-vous bien remarqué les mots (***else if***) et (***else***) dans le bloc de codes précédent ? Si la première condition échoue le test, alors le code va regarder pour la commande : **sinon** (***else***). Si celle-ci est suivie d'une nouvelle condition (***if***), elle sera analysée. Si toutes les conditions échouent, le code exécute alors les instructions dans la condition (***else***).

Notez-bien!

Si vous avez plus qu'une instruction, les accolades sont encore une fois nécessaires pour chaque condition.

Vous pouvez également évaluer plusieurs conditions dans une seule instruction avec l'opérateur : **et** (***&&***).

```
var age = 21;
var sexe = "f";
if (age >= 21 && sexe == "f")
      Debug.Log ("Femme majeure");
```

Dans le dernier exemple, vous venez de voir les opérateurs **plus grand ou égal** (**>=**) **et** (***&&***). Si les deux conditions sont vraies, ses commandes seront alors exécutées. Par contre, si une seule est fausse, l'instruction sera invalide et ses commandes ne seront pas exécutées.

Vous venez d'apprendre comment évaluer deux conditions simultanément avec : ***et*** mais il existe également l'opérateur : ***ou*** (||). Reprenons notre dernier exemple, disons que le sexe d'un usager peut seulement être un caractère soit **m** ou **f**. Par contre, un usager peut également écrire : **M** ou **F** en majuscule car JavaScript est sensible aux majuscules et aux minuscules. Alors, si vous voulez vous assurer que **F** et **f** donnent les mêmes résultats, vous devez accepter que sexe soit l'une ou l'autre de ces deux réponses :

```
var age = 21;
var sexe = "f";
if (age >= 21 && (sexe == "f" || sexe == "F"))
      Debug.Log ("Femme majeure");
```

Maintenant, pour que la condition soit vraie, il faut que l'âge soit 21 ou plus ***et*** que le sexe soit **F *ou* f.** Si l'âge est valide et que l'une des deux réponses pour sexe est vraie, ses commandes seront exécutées. Par contre, si sexe n'est pas une de ces deux réponses, la condition aura échoué.

Lorsque vous avez plusieurs valeurs à évaluer dans un tableau (***Array***), vous pouvez vous servir de l'opération ***dans*** (***in***) pour rechercher s'il existe une valeur quelque part.

```
var noms = ["Marc","Louis","France","Christine"];
if ("Louis" in noms)
      Debug.Log ("Louis est présent");
```

Si une variable peut prendre plusieurs valeurs différentes et que vous voulez toutes les évaluer individuellement, vous pouvez vous servir des instructions (***switch***) et (***case***) pour évaluer toutes les possibilités :

```
var groupe = "hirondelles";
switch (groupe){
      case "geais bleus":
         Debug.Log ("Visite à 12:00");
      break;
      case "hirondelles":
         Debug.Log ("Visite à 13:00");
      break;
      case "canards":
         Debug.Log ("Visite à 14:00");
      break;
      default:
         Debug.Log ("groupe non valide");
      break;
}
```

Dans l'exemple ci-dessus, vous avez testé la valeur d'un texte (***String***) avec différentes valeurs afin de trouver celle qui était appropriée au groupe. Vous avez probablement remarqué la commande (***break***). Lorsque cette instruction est rencontrée, l'exécution du bloc de codes est interrompue. Ceci est un moyen de délimiter les commandes pour les différents cas. De plus, il y a l'instruction (***default***) qui est l'équivalent du ***sinon*** (***else***) et qui s'exécute seulement lorsque tous les autres cas ont échoué.

Les boucles

Les boucles vous permettent de répéter certaines commandes, un nombre indéterminé de fois jusqu'à ce qu'une condition soit remplie.

Prenez par exemple :

```
var nombre = 0;
nombre += 10;
Debug.Log(nombre);
nombre += 10;
Debug.Log(nombre);
nombre += 10;
Debug.Log(nombre);
nombre += 10;
Debug.Log(nombre);
```

Ce code est très redondant, il est alors préférable d'utiliser une boucle. Il existe trois types de boucles : **Pour** (***for***), **tant que** (***while***), **fait…tant que** (***do… while***). Vous allez débuter par la boucle : **pour.**

Par exemple, vous pouvez écrire le bloc de codes précédent comme ceci :

```
var nombre = 0;
for (var i=0; i<4; i++){
      nombre += 10;
      Debug.Log(nombre);
}
```

En détails…

La boucle : **pour** (***for***) a besoin de **3** paramètres : un compteur initialisé : (***var*** **i=0**), une condition (boucle tant que celle-ci est vraie) **i<4** et l'incrémentation du compteur **i++**. Les paramètres sont séparés par des points-virgules (**;**). Dans l'exemple précédent voilà comment le code a interprété cette boucle :

1. *D'abord, la variable **nombre** est crée et = 0.*
2. *La variable **i** est crée et (i= 0).*
3. *nombre + 10 (nombre=10).*
4. ***On affiche 10 dans le débogueur.***
5. *i++ (i=1).*

6. *nombre + 10 (nombre=20).*
7. ***On affiche 20 dans le débogueur.***
8. *i++ (i=2).*
9. *nombre + 10 (nombre=30).*
10. ***On affiche 30 dans le débogueur.***
11. *i++ (i=3).*
12. *nombre + 10 (nombre=40).*
13. ***On affiche 40 dans le débogueur.***
14. *i++ (i=4). La condition (i<4) est **fausse**, on quitte la boucle.*

Vous pouvez également utiliser la valeur du compteur à l'intérieur de la boucle. Dans l'exemple précédent, vous augmentez la valeur d'un nombre de **10** seulement pour afficher sa valeur dans le débogueur. Vous auriez le même résultat en écrivant :

```
for (var i=10; i<=40; i+=10){
      Debug.Log(i);
}
```

En détails...

Vous voyez qu'en changeant la valeur d'initialisation de mon compteur **var i=10**, de la condition d'exécution **i<=40** et de l'incrémentation **i+=10,** j'obtiens le même résultat mais avec une variable et une instruction en moins.

Non seulement on peut augmenter l'incrémentation par un nombre plus grand que 1, mais on peut également compter à l'envers.

```
for (var i=4; i>0; i--){
      Debug.Log(i);
}
```

Notez-bien!

Vous n'avez pas à utiliser **i** *comme **variable**, vous pouvez nommer celle-ci comme vous voulez. Également, vous pouvez utiliser une variable existante au lieu d'en déclarer une dans l'instruction.*

Lorsque vous ne connaissez pas le nombre de fois qu'une boucle doit s'exécuter avant de s'arrêter, vous utilisez le compteur de type : **tant que** (***while***). **Tant que** la condition est vraie, on exécute ses instructions. Il est possible

que la condition soit fausse avant même d'avoir exécuté une fois la boucle, c'est le principal avantage de ce type de boucle.

```
var nombre = 0;
while (nombre < 10){
      nombre++;
      Debug.Log(nombre);
}
```

Notez-bien!

C'est audacieux d'utiliser ce type de boucle, car il n'y a pas de compteur ni d'incrémentation automatique pour s'assurer de ne pas tomber dans une boucle infinie (ce qui risque de faire planter l'application). Vous devez penser à initialiser manuellement une variable avant l'exécution de la boucle ainsi qu'une incrémentation dans les instructions de la boucle.

Finalement, si vous savez qu'une boucle doit s'exécuter au moins une fois avant d'être validée, vous pouvez utiliser : ***fait…tant que*** (**do…while**).

```
var noms = ["Marc","Louis","France","Christine"];
var present = false;
var i=0;
do{
      if (nom[i]=="France")
      present = true;
      i++;
}while (!present || i<noms.Length);
```

En détails…

Précédemment, vous avez fait les instructions qui viennent après (**do**) sans avoir validé quoi que ce soit. Quand vous arrivez à la condition, vous devez en utiliser deux pour sortir de la boucle ou un booléen quand vous trouvez ce que vous cherchez. Vous pouvez aussi utiliser un compteur qui vous permet de sortir si vous avez validé tous les éléments sans jamais avoir trouvé ce que vous cherchez. Il est possible d'entrer dans une boucle sans fin, si l'on n'est pas prudent avec celle-ci. Quant à **noms.length,** c'est une propriété qui est intégrée aux **tableaux** (**Arrays**), elle permet de retrouver le nombre d'éléments dans la variable.

Nous reviendrons un peu plus loin dans ce chapitre sur les propriétés intégrées. Mais avant, le présent paragraphe parlera des fonctions.

Les fonctions

Une fonction (**function**) est un bloc de codes réutilisables que l'on exécute lors d'une seule commande. Vous allez débuter par une fonction simple qui affiche : **Bonjour le monde** dans la console.

```
function DitBonjour(){
      Debug.Log("Bonjour le monde");
}
```

Pour exécuter la fonction, on l'appelle comme ceci :

```
DitBonjour();
```

Notez-bien!

*Remarquez qu'on écrit des parenthèses **( )** après notre fonction. Celles-ci sont requises quand on appelle et définit une fonction. Remarquez également l'écriture de la fonction avec une lettre majuscule au début. Ce n'est pas nécessaire, mais dans Unity, c'est une convention d'écrire les fonctions avec une lettre majuscule et les variables avec une lettre minuscule, afin de les distinguer.*

Maintenant, si l'on veut rendre cette fonction plus versatile et afficher un message personnalisé, au lieu de : **bonjour le monde**, on doit alors passer un paramètre à la fonction.

```
function Dit(texte:String){
      Debug.Log(texte);
}
```

Dans l'exemple précédent, « texte » est une variable temporaire qui peut seulement être utilisée dans la fonction.

Maintenant, lorsque vous voulez appeler la **fonction**, vous devez passer un argument en **paramètre**.

Notez-bien!

Remarquez que dans l'exemple précédent, il est obligatoire de définir explicitement le type de paramètre à utiliser. Dans ce cas, « texte » est un ***String.***

```
Dit("Unity c'est top!");
```

Vous venez de passer un bloc de texte brut, mais vous pouvez également utiliser une ***variable*** comme **paramètre**.

```
var monTexte = "J'aime Unity";
Dit(monTexte);
```

Non seulement une fonction peut recevoir un argument, mais elle peut également nous retourner une variable avec (***return***). La fonction suivante vérifie si un nombre est pair et retourne **vrai** (***true***) si c'est le cas, ou **faux** (***false***)

```
function NombrePair(nombre:int){
      if (number % 2 == 0)
          return (true);
      else
          return (false);
}

var chiffre = 10;
if (NombrePair(chiffre)){
     Debug.Log("Le nombre: "+chiffre+" est pair");
}
```

Quand la commande (***return***) est exécutée, la fonction est automatiquement terminée. Le reste de la fonction est **ignoré**. La commande (***return***) n'est pas obligée de retourner quoi que ce soit.

```
function Update(){
 if (!enCours) return; //Quitte la fonction
}
```

La fonction ***Update()*** est l'une des principales d'Unity. Vous n'avez pas à appeler cette fonction, elle l'est automatiquement à chaque image. Il existe plusieurs autres fonctions qu'Unity reconnaît et utilise nativement, selon les circonstances.

En voici quelques exemples :

fonction	Description
Start()	Cette fonction est appelée au lancement du script avant le premier ***Update()***, elle s'exécute seulement si le ***Component* du script est actif**.
Update()	Cette fonction est appelée à chaque image.

Les fonctions précédentes sont ajoutées automatiquement lorsqu'un nouveau script est créé, mais il existe plusieurs fonctions qu'Unity reconnaît nativement et qui sont cachées par défaut.

fonction	Description
Awake()	Cette fonction est appelée au lancement du script, ***avant Start()*** et celle-ci est appelée même si le ***Component*** du script n'est pas actif.
FixedUpdate()	Cette fonction est appelée à chaque image fixe. Il est recommandé d'utiliser cette fonction au lieu d'***Update(),*** quand on manipule un ***rigidBody.***
LateUpdate()	Cette fonction est appelée après tous les autres types d'***Update() de la scène***. C'est idéal quand on veut ordonner l'ordre d'exécution des scripts.
OnGUI()	Cette fonction sert pour les éléments de l'interface graphique, celle-ci est appelée indépendamment d'***Update()*** même si le jeu est en pause.(**deltaTime ==0**).

Les listes précédentes sont utilisées pour tous les scripts et pour tous les objets. La liste suivante est spécifique aux ***colliders*** et aux ***colliders 2D*** *(en mode 2D seulement).*

fonction	Description
OnCollisionEnter()/ OnCollisionEnter2D()	Cette fonction est appelée quand un ***collider/rigidBody*** commence à toucher un second ***rigidBody/collider.***
OnCollisionStay()/ OnCollisionStay2D()	Cette fonction est appelée à chaque image pour chaque ***rigidBody/collider*** qui se touche.
OnCollisionExit()/ OnCollisionExit2D()	Cette fonction est appelée quand un ***collider/rigidBody*** arrête de toucher un second ***rigidBody/collider.***

Une liste de fonctions similaires existe pour un Trigger et un Trigger***2D*** *(en mode 2D seulement).*

fonction	Description
OnTriggerEnter()/ OnTriggerEnter2D()	Cette fonction est appelée quand un ***collider*** entre dans la zone d'un Trigger***.***
OnTriggerStay()/ OnTriggerStay2D()	Cette fonction est appelée à chaque image quand un ***collider*** est dans la zone d'un Trigger***.***
OnTriggerExit()/ OnTriggerExit2D()	Cette fonction est appelée quand un ***collider*** sort de la zone d'un Trigger***.***

Il existe également des fonctions durant la création, l'activation et la destruction d'objets :

fonction	Description
OnEnable()	Cette fonction est appelée quand un ***objet*** devient **visible** et **actif**.
OnDisable()	Cette fonction est appelée quand un objet devient **inactif** ou **désactivé**.
OnDestroy()	Cette fonction est appelée quand l'objet est sur le point d'être supprimé.
OnBecameVisible()	Cette fonction est appelée quand le ***renderer*** d'un objet devient visible par une ***caméra***.
OnBecameInvisible()	Cette fonction est appelée quand le ***renderer*** d'un objet n'est plus visible par les ***caméras***.
OnApplicationPause()	Cette fonction est appelée sur tous les scripts quand le jeu est en ***pause.***
OnApplicationQuit()	Cette fonction est appelée sur tous les scripts quand le jeu est sur le point d'être ***fermé***.
OnApplicationFocus()	Cette fonction est appelée sur tous les scripts quand le jeu **reçoit** ou **perd** le ***focus***.

Il existe plusieurs autres fonctions d'Unity. Pour plus de détails, veuillez consulter la documentation officielle de **MonoBehaviour** :

La surcharge de fonctions (surdéfinition)

Une fonction peut être polymorphique, c'est-à-dire qu'il peut y avoir différentes versions d'une fonction avec des arguments différents.

Comme mentionné précédemment, quand on passe un paramètre à une fonction, il faut explicitement définir son type : ***String***, ***boolean***, ***int***, ***float*****…** Du même coup, on peut définir différents types de variables pour les différentes versions de la fonction.

```
function ChangePos(coorXYZ:Vector3){
 transform.position = coorXYZ;
}

function ChangePos(coorX:float, coorY:float, coorZ:float){
 transform.position = Vector3 (coorX, coorY, coorZ);
}

function ChangePos(objRef:Transform){
 transform.position = objRef.position;
}
```

En détails...

Dans l'exemple précédent, la fonction **ChangePos** a été définie trois fois avec trois arguments différents. Le rôle de cette fonction est de repositionner un objet dans la scène de trois façons différentes, on peut appeler cette fonction comme ceci:

```
var PositionVecteurs = Vector3(10.0,10.0,10.0);
var objectRef : Transform;
/*Ici je dois assigner un objet manuellement dans l'éditeur*/

ChangePos(PositionVecteurs);
ChangePos(10.0,10.0,10.0);
ChangePos(objetRef);
```

Les fonctions et les variables intégrées

Vous pouvez accéder a une fonction ou une variable définie dans un type d'élément, en utilisant un point (**.**) et le nom de l'élément auquel on veut accéder.

Dans un exercice précédent, nous avons utilisé la propriété (***Length***) pour obtenir le nombre de valeurs dans un tableau (***Array***) ou le nombre de lettres dans un mot.

```
var noms = ["Marc","Louis","France","Christine"];
var total = noms.Length;
```

Vous pouvez également utiliser (***Length***) dans une boucle :

```
for (var i=0;i<noms.Length;i++){
 Debug.Log(noms[i]) ;
}
```

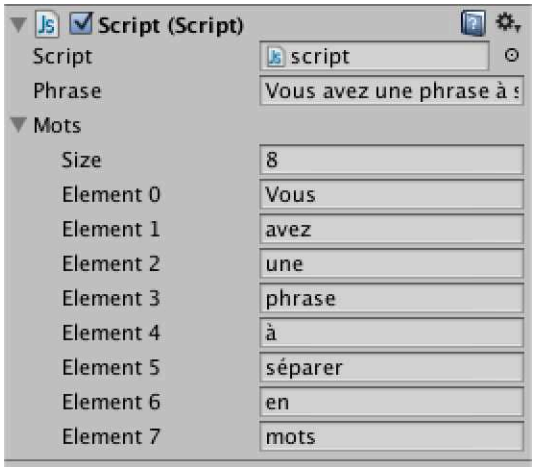
Même dans le cas: ***Debug.Log()***, quand on utilise ***Log()***, on appelle une fonction intégrée à (***Debug***) qui sert à afficher du contenu dans la console.

D'autres fonctions vont jouer un rôle plus important dans vos scripts. Par exemple, la fonction : ***ToString()*** peut servir à convertir un nombre en texte :

```
var nombre :int = 100;
var texte :String = nombre.ToString();
```

Pour séparer un texte (***String***) en tableau (***Array***), on utilise la fonction ***Split()*** :

```
var phrase = "Vous avez une phrase à séparer en mots";
var mots = phrase.Split(" "[0]);
Debug.Log(mots.Length);
```



Il existe plusieurs fonctions et variables intégrées pour chaque type d'élément dans Unity. Pour en connaître davantage, veuillez consulter la documentation officielle :

http://docs.unity3d.com

Les classes

Les variables ont différents types comme : ***String, int, float, boolean…*** Mais si vous voulez créer votre propre type d'objet qui effectue quelque chose de différent, vous devez vous servir des ***classes***.

Avant de créer une nouvelle **classe**, vous devez absolument nommer votre fichier script du même nom que la **classe**. Pour l'exemple suivant, le fichier doit absolument se nommer **Personne.js** pour fonctionner :

```
class Personne {
 var nom : String;
 var carriere : String;
}
```

Maintenant, si l'on veut créer une : ***Personne, dans un autre script.***

```
var phil: Personne();
phil.nom = "Philippe Lebond";
phil.carriere = "professeur";
Debug.Log(phil.nom + "est un " + phil.carriere);
```

En détails…

La **classe** précédente possède deux variables (ou paramètres) internes : **nom** et **carrière**. Vous pouvez accéder à ces paramètres simplement en tapant le nom de votre instance (dans notre exemple : **phil**) suivie par un point et le nom de la propriété.

Vous pouvez également passer des arguments quand vous créez une instance de la **classe**. Vous le faites avec ce que l'on appelle un **constructeur**, une fonction spéciale qui est automatiquement appelée, quand une copie de la **classe** est créée. Cette fonction porte le même nom que votre classe et on l'utilise pour initialiser la **classe**.

```
class Animal {
  var nom : String;
  function Animal(valeur :String){
      nom = valeur;
      Debug.Log(nom+" est créé");
  }
}
```

Pour créer un nouvel animal, on le déclare comme ceci :

```
var chien = Animal("Rex");
```

Les ***classes*** ont également des **fonctions** régulières appelées **méthodes** (ou ***methods***). Vous appelez ces **méthodes** de la même façon que pour accéder un argument, avec un point **(.)** et le nom de la **fonction ().**

```
class Personne {
  var nom : String;
  function Personne(val : String){
     nom = val;
  }
  function Parle(qui : Personne){
     Debug.Log(nom + " parle à "+ qui.nom);
  }
}
```

Avec les fonctions suivantes, vous pouvez créer :

```
var mario = Personne("Mario");
var phil = Personne("Phil");

mario.Parle(phil);
```

Vous avez vu la base des classes, nous allons dorénavant augmenter le niveau de difficulté.

L'héritage

Une classe peut hériter des fonctionnalités d'une autre. Les classes de base qui sont utilisées pour d'autres classes sont nommées classe (***parent***). Celle qui hérite est également référée à une classe **dérivée**.

Le bloc de codes suivant sera notre **classe de base** :

```
class Personne {
   var nom : String;
   function Personne(val : String){
      nom = val;
   }
   function Parle(qui : Personne){
      Debug.Log(nom + " parle à "+ qui.nom);
   }
   function Marche(){
      Debug.Log(nom + " marche ");
   }
}
```

Vous allez maintenant créer une **classe dérivée,** avec l'instruction : (***extends***).

```
class Enfant extends Personne {
   var age: int;
   function Enfant(n: String, a : int){
     super(n);
     age = a;
   }
   function Parle(qui : Personne){
      super.Parle(qui);
      Debug.Log(" de ses jouets!");
   }
   function Joue(){
      Debug.Log(nom + " joue");
   }
}
```

Notez-bien!

Quand vous créez une classe dérivée, vous pouvez appeler les fonctions et les variables de la classe parent, en utilisant l'instruction : ***super.***

Vous pouvez maintenant créer un nouvel enfant qui pourra faire tout ce qu'une personne peut accomplir ainsi que des fonctions et paramètres propres à cette nouvelle **classe**. Quand on appelle la fonction **Parle**(), on appelle la fonction de l'enfant et du parent. Lorsqu'on appelle la fonction **Joue()**, on appelle uniquement la fonction de l'enfant. Finalement, quand on appelle **Marche()**, on appelle la fonction du parent, car celle-ci n'est pas redéfinie dans le script de l'enfant.

```
var paul = Enfant("Paul", 14);
paul.Joue();
paul.Marche();
```

Maintenant que vous avez vu les bases de la programmation JavaScript, vous devriez être en mesure de programmer tous les exercices qui vont suivre !

unity

Chapitre 4 - Programmation de Rebonds : partie 2

Matière couverte dans ce chapitre

- **Déplacement d'une plateforme avec la souris**
- **Déplacement de la balle avec un facteur aléatoire**
- **Création des règles du jeu**
- **Système de particules**
- **Habillage sonore**
- **Exportation d'un exécutable**

Dans ce chapitre, vous verrez comment programmer et exporter l'exercice **Rebonds** que vous avez débuté au chapitre 2. À la fin de ce chapitre, vous obtiendrez un jeu indépendant pour **Windows**, **OS X** ou **Linux** (c'est à votre discrétion). De plus, nous aborderons les théories sur la création de particules, l'habillage sonore de base ainsi que le gestionnaire ***Build settings*** qui permet de rassembler les différentes scènes de votre jeu et de l'empaqueter pour les différentes plateformes.

Atelier pratique

Programmation de Rebonds

Débutons avec un script pour bouger la palette avec le mouvement de la souris.

Déplacer la plateforme

1. Dans un espace libre du panneau *Projet*, faites un clic secondaire.

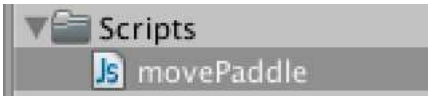
➢ **Create > Folder**

1.1. Nommez ce répertoire : **Scripts.**

2. Sur le répertoire **Scripts**, faites un clic secondaire puis :

➢ **Create > Javascript**

2.1. Nommez le script : **movePaddle.**

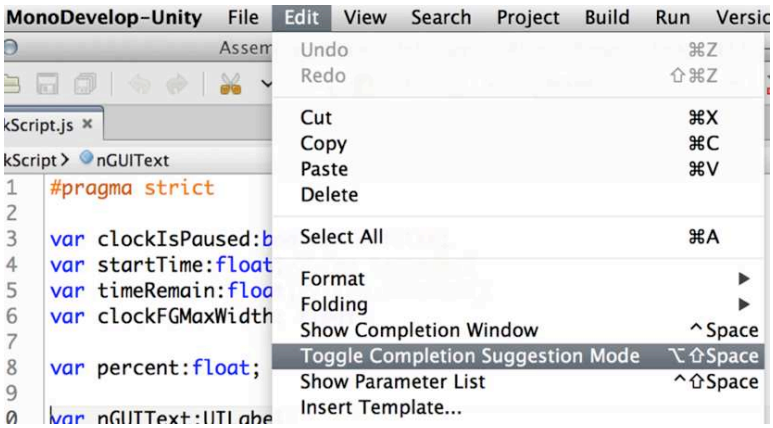


Ouvrez ce fichier (avec *MonoDevelop* ou un autre éditeur de script).

Avant de débuter, si vous utilisez *MonoDevelop*, pensez à désactiver l'auto-remplissage des commandes (***Toggle Completion Suggestion Mode***), car il nuit plus souvent qu'il aide.

3. À partir de la barre de menus de *MonoDevelop,* appuyez sur :

Edit | Toggle Completion Suggestion Mode.



Maintenant, ajoutez ce bloc de codes :

```
var speed = 3.0;

function Update () {
    var trans_Y : float = Input.GetAxis ("Mouse Y") * speed;
    var trans_X : float = Input.GetAxis ("Mouse X") * speed;
    trans_Y = trans_Y * Time.deltaTime;
    trans_X = trans_X * Time.deltaTime;
    transform.Translate (trans_X, 0, trans_Y);
}
```

↘ *Les textes en caractères **gras** sont à ajouter, le reste demeure inchangé.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Ce petit script reçoit les valeurs des axes de la souris (***Input.GexAxis (Mouse Y) et Mouse X***), multiplie ces valeurs par la vitesse (***speed = 3.0***) et conserve le résultat dans les variables (***trans_Y*** *et* ***trans_X***).

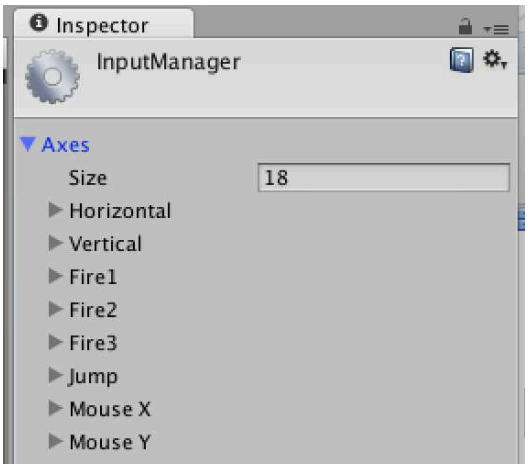
Ensuite, celui-ci multiplie ces chiffres par une variable système (***Time.deltaTime***) afin d'ajuster la vitesse de déplacement pour un intervalle de temps (***Update()***).

Finalement, le script déplace la plateforme sur l'axe X et Z avec l'instruction : (***transform.Translate(valX, 0, valZ)***).

Notez-bien!

Input.GetAxis *se réfère au* ***Input Manager****, vous pouvez ajouter, changer et retirer des* ***Axes*** *à partir de la barre de menus :*

Edit | Project settings | Input



Vous allez à présent appliquer le script que vous venez de créer sur la plateforme.

4. Glissez ce script sur l'objet **PlayerPlat** dans le panneau *Hierarchy*.

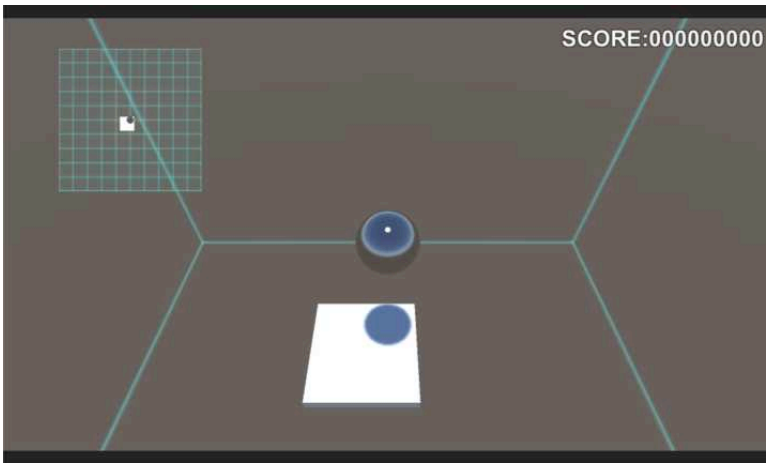
Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

La palette devrait se déplacer librement, mais le curseur est nuisible. On aimerait s'en passer.

Nous y reviendrons plus tard.

Un autre détail à remarquer, la caméra ne suit pas la plateforme.

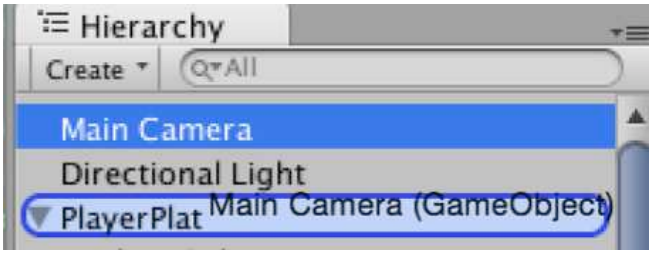


Notez-bien!

*Si vous voulez modifier maintenant la vitesse de votre palette, vous devez passer par **l'Inspecteur** car celui-ci va mémoriser la dernière valeur reçue, même si l'on change la valeur, par défaut, dans le code.*

Vous allez fixer la caméra simplement en changeant la parenté de **Main Camera.**

Retournez en mode édition.



5.Glissez Main Camera du panneau *Hierarchy* sur l'objet **PlayerPlat**.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

La caméra devrait coller en place, mais la vue serait meilleure de plus haut.

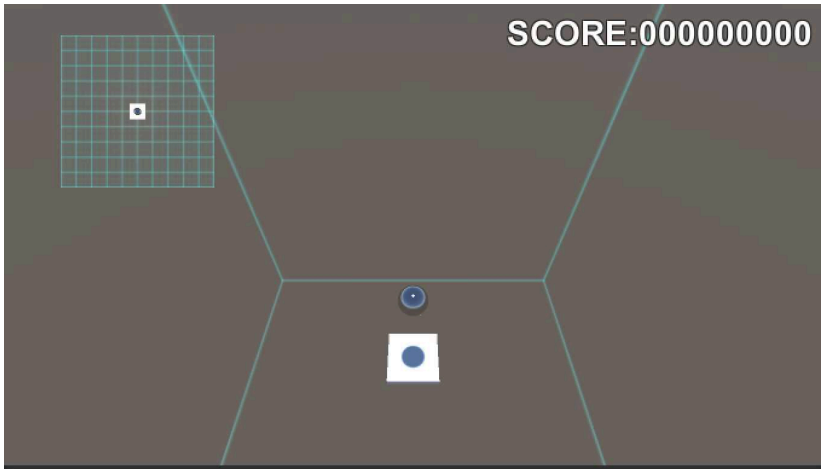
Retournez en mode édition.

6. Sélectionnez **Main Camera** dans votre *Hierarchy*.

6.1. Dans l'*Inspector,* changez :

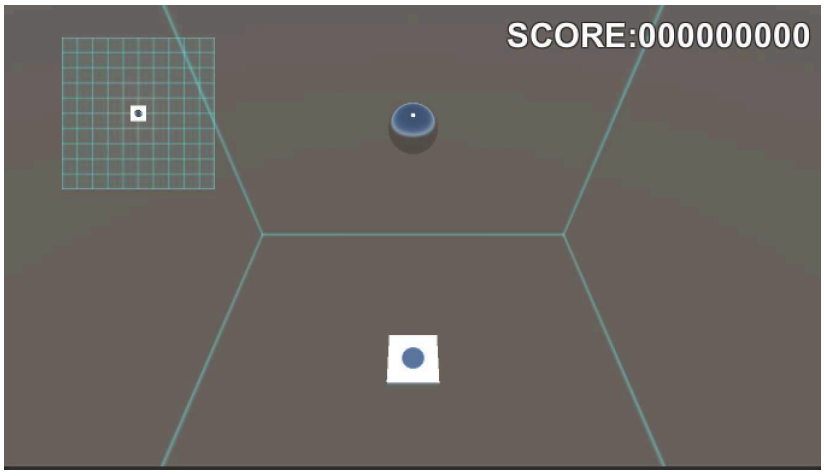
Transform
6.1.1. ***Position*** **: X=** 0, **Y=** 67, **Z=** -3.2
6.1.2. ***FOV*** : 67

Le point de vue devrait être similaire à ceci :



Manipuler la balle

7. Sélectionnez **BalleRebond** dans votre *Hierarchy*.



7.1.Dans l'*Inspector,* changez la valeur de :

Transform
7.1.1. ***Position :*** **Y=** 5

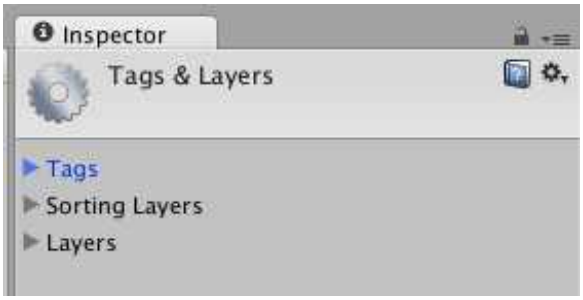
Pour le *gameplay*, il est préférable de repositionner **worldLimit.**

8. Sélectionnez **worldLimit.**

8.1. Dans l'*Inspector,* changez :

Transform
8.1.1. ***Position :*** **Y=** -1.45

Avant d'aller plus loin, vous aurez besoin d'un nouveau ***Tag*** pour la balle, afin de différencier la palette du reste du jeu. Cette action vous permettra de réagir différemment selon que vous touchez la plateforme ou les murs.



9. Dans la barre de menus, allez dans :

Edit | Project Settings | Tags and Layers

Dans *l'Inspector,* vous devriez voir 3 catégories : ***Tags***, ***Sorting Layers*** et ***Layers***.

10. Appuyez sur la flèche à gauche de ***Tags.***

10.1. Appuyez sur le **(+)** afin d'ajouter un nouveau champ.

10.2. Ajoutez le ***Tag*** : **paddle** à l'élément ***Tag 0***.

11. Dans votre *Hierarchy*, sélectionnez l'objet : **PlayerPlat.**

11.1. Assignez le nouveau ***Tag*** : **paddle** à l'objet.

Pour le challenge, comme la boîte de collisions est plate, la balle rebondit toujours en ligne droite et va même jusqu'à gagner de la vélocité. Vous allez faire un script qui gère la façon dont la balle va rebondir ainsi que les différentes collisions avec la scène.

12. Dans le panneau *Projet*, faites un clic secondaire sur le répertoire **Scripts,** puis :

- **Create > Javascript**

12.1. Nommez le script : **gameRules.**

Ouvrez le script « gameRules.js ».

Ajoutez la fonction suivante :

```
function OnCollisionEnter(collision : Collision) {
  var colPos = collision.transform.position ;
  var rigid = GetComponent.<Rigidbody>();
  if (collision.collider.tag == "paddle"){
    var diffX = colPos.x - transform.position.x;
    var diffZ = colPos.z - transform.position.z;
    rigid.velocity = Vector3(-diffX, 8, -diffZ);
    Debug.Log("x: "+ diffX +" z: "+ diffZ);
  }
}
```

Sauvegardez le script et retournez dans l'éditeur.

En détails…

La fonction (**OnCollisionEnter()**) est appelée automatiquement quand une collision survient. Si le Tag de la plateforme **paddle** est présent, on calcule alors la différence entre le centre de la balle et le centre de la plateforme (**collision.transform.position.x–transform.position.x**, la même chose pour **Position.z**) ensuite, on place le résultat dans les variables (**diffX** et **diffZ**).

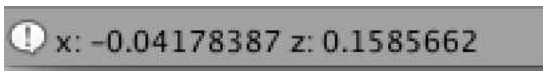
Finalement, on applique une force sur les **axes x** et **z** de la balle, pour la faire rebondir dans la direction opposée (**rigid.velocity = Vector3(-diffX, 8, -diffZ)**), la valeur de **Y= 8** pour la faire rebondir vers le haut.

13. Appliquez le script **gameRules** sur le ***GameObject*** : **BalleRebond**.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

La balle devrait bouger d'une façon similaire au classique : **PONG**, sauf qu'un joueur peut facilement tricher en ne touchant à rien et en laissant la plateforme parfaitement alignée avec la balle.



Dans la console *Debug*, on peut observer la valeur de la **Position X** et **Z** qui sépare la balle du centre de la palette.

Retournez en mode édition.

Ajoutez maintenant un élément de hasard dans le déplacement de la balle.

Utilisation des valeurs aléatoires (Random)

Ouvrez le script « gameRules.js ».

Ajoutez la fonction suivante :

```
var minTranslation = 0.05;

function OnCollisionEnter(collision : Collision) {
    var colPos = collision.transform.position ;
    var rigid = GetComponent.<Rigidbody>();
    if (collision.collider.tag == "paddle"){
        var diffX = minTranslation+(colPos.x -
                            transform.position.x)*Random.Range(1,10);
        var diffZ = minTranslation+(colPos.z -
                            transform.position.z)*Random.Range(1,10);
        rigid.velocity = Vector3(-diffX, 8, -diffZ);
        Debug.Log("x: "+diffX+" z: "+diffZ);
    }
}
```

↘ *Les textes en **gras** sont à ajouter ou ~~**retirer**~~ le reste demeure inchangé.*

Sauvegardez le script et retournez dans l'éditeur.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

Il est désormais plus difficile de maintenir la balle parfaitement au centre de la plateforme.

Retournez en mode édition.

La prochaine étape consistera à ajouter une façon de marquer des points et d'augmenter la difficulté en utilisant un dernier script pour cet exercice.

En détails...

Ces petits changements comprennent une valeur de déplacement minimale de 0.05 que l'on additionne aux variables (**diffX** et **diffZ**) avant de multiplier ces résultats par une valeur aléatoire entre 1 et 10 (**Random.Range(1,10)**).

Finalement, on retire la ligne qui affiche la valeur des variables (**diffX** et **diffZ**) dans le débogueur.

Garder la trace du pointage

Ouvrez le script « gameRules.js ».

Ajoutez le code suivant :

```
var minTranslation = 0.05;

var scoreboard = 0; //Pointage en chiffres.
var preTextScore = "SCORE: "; //Texte permanent.
var scoreTx = "0"; //Pointage en lettres.
var pointsPerBounce = 35; //Valeur d'un bond.
var scoreGUI:UI.Text; //Champ de texte

function addScore(){
  scoreboard += pointsPerBounce;
  scoreTx = scoreboard.ToString();
}

function OnGUI(){
  scoreGUI.text = preTextScore + scoreTx;
}
…
```

*Les textes en **gras** sont à ajouter, les **commentaires (//)** sont optionnels mais ils aident à comprendre le code et « … » est utilisé pour simplifier la lecture.*

En détails…

Les variables sont expliquées dans les commentaires du script et la fonction **addScore()** effectue deux actions : elle ajoute 35 points à la variable scoreboard et convertit le nombre en texte. La fonction **OnGUI()** rafraîchit le texte à l'écran, en combinant le texte permanent et le pointage en texte.

Vous devez également ajouter une ligne de codes dans le script **gameRules** quand vous frappez l'objet : **paddle**.

```
…
function OnCollisionEnter(collision : Collision) {
    var colPos = collision.transform.position ;
    var rigid = GetComponent.<Rigidbody>();
    if (collision.collider.tag == "paddle"){
        var diffX = minTranslation+(colPos.x -
                        transform.position.x)*Random.Range(1,10);
        var diffZ = minTranslation+(colPos.z -
                        transform.position.z)*Random.Range(1,10);
        rigid.velocity = Vector3(-diffX, 8, -diffZ);
        addScore();
    }
}
```

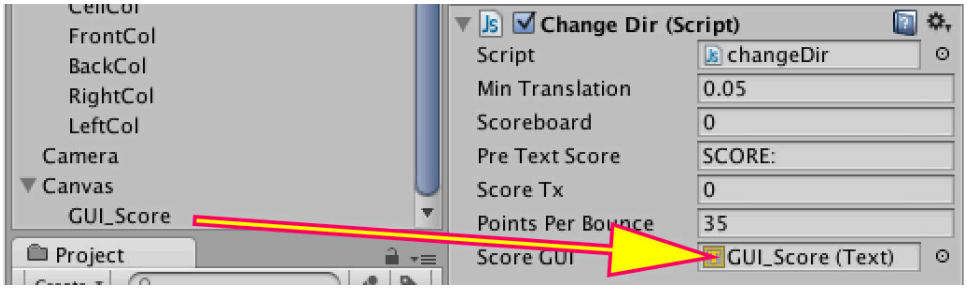
*Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

La fonction (**addScore()**) est appelée à chaque bond, au moment d'une collision.

14. Sélectionnez l'objet : **BalleRebond** dans votre *Hierarchy,* puis :

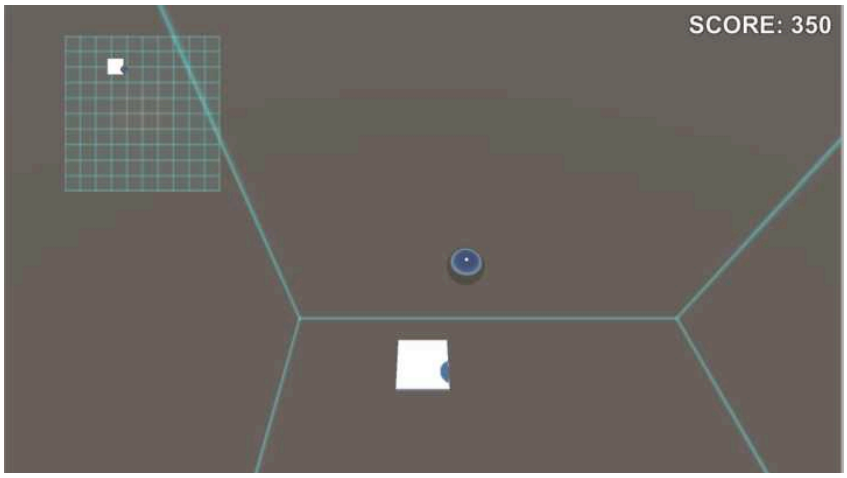


14.1. Dans *l'Inspector*, glissez **GUI_Score** provenant du **Canvas** dans la variable **score GUI**.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

Le pointage devrait maintenant augmenter : 35, 70…



Retournez en mode édition.

Augmentation de la difficulté

Maintenant, voici un challenge de plus, vous allez augmenter la difficulté après chaque bond.

Ouvrez le script « gameRules.js ».

Ajoutez le code suivant :

```
function addScore(){
      scoreboard = scoreboard +pointsPerBounce;
      scoreTx = scoreboard.ToString();
      increaseDiff();
}

function increaseDiff(){
     var timeScaleLimit:float = 3;
     var timeScaler:float = Time.timeScale + 0.03;
     if (timeScaler > timeScaleLimit)
        timeScaler = timeScaleLimit;
     Time.timeScale = timeScaler;
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé.*

Sauvegardez le script et retournez dans l'éditeur.

En détails…

La fonction (**increaseDiff()**) augmente la vitesse globale du jeu (**time.timeScale**) jusqu'à ce que cette valeur soit égale à la valeur de (**timeScaleLimit = 3**).

Cette nouvelle fonction est appelée à chaque fois qu'un point est marqué, grâce à la fonction (**addScore()**).

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

Le jeu fonctionne, mais plus vous marquez de points, plus la palette devient sensible. Vous devez donc ajouter un petit calcul pour la vitesse de déplacement du curseur, qui dépend du ***TimeScale***.

Retournez en mode édition.

Ouvrez le script « movePaddle.js ».

Ajoutez le code suivant :

```
var speed = 3.0;
function Update () {
    var trans_Y : float = Input.GetAxis ("Mouse Y") * speed;
    var trans_X : float = Input.GetAxis ("Mouse X") * speed;
    trans_Y = trans_Y * (Time.deltaTime/Time.timeScale);
    trans_X = trans_X *(Time.deltaTime/Time.timeScale);
    transform.Translate (trans_X, 0, trans_Y);
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé.*

Sauvegardez le script et retournez dans l'éditeur.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

La plateforme devrait garder la même sensibilité, tout au long du jeu.

Retournez en mode édition.

Effets de particules

Ajoutez maintenant un effet pour signaler la fin ; un moyen évident qui expliquera que le jeu est terminé.

Pour ce faire, vous allez créer un effet de dissolution pour la balle lorsqu'elle touche à un autre élément que la palette.

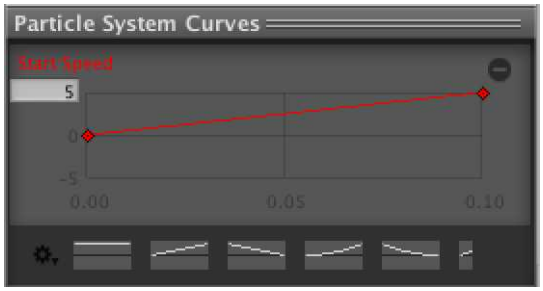
15. À partir de la barre de menus, cliquez sur :

GameObject | Particles System

15.1. Nommez l'émetteur : **Smoke.**

15.2. Dans *l'Inspector*, changez les valeurs des paramètres suivants :

Paramètres de bases
15.2.1. ***Duration*** : 0.10
15.2.2. ***Looping*** : non
15.2.3. ***Start Lifetime*** : 0.5
15.2.4. ***Start Speed*** : Cliquez sur la flèche au bout du champ et changez **Constant** pour **Curve.**

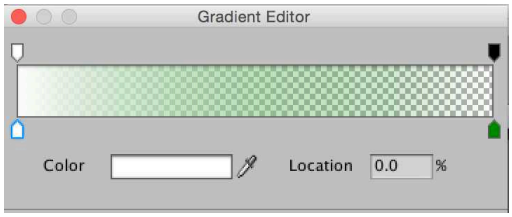


Changez le type de courbe pour une **ligne ascendante**.

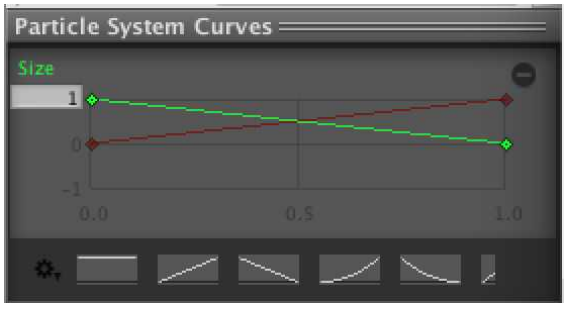
15.2.5. ***Start Size*** **:** 1
15.2.6. ***Simulation Space*** **:** World
15.2.7. ***Play On Awake*** **:** non
15.2.8. ***Max Particles*** **:** 1000

Emission
15.2.9. ***Rate*** : 300
15.2.10. ***Time***

Shape
15.2.11. ***Shape*** *: Sphere*
15.2.12. ***Radius*** : 0.5
15.2.13. ***Random Direction*** *: oui*



Color over lifetime
15.2.14.Dégradé : **Blanc**, alpha : 100% à **Vert**, alpha : 0%



Size over lifetime
Changez le type de courbe pour une **ligne descendante.**

Vous allez maintenant ajouter un cercle de lumière (***Halo***) autour de la balle afin de donner un effet phosphorescent.

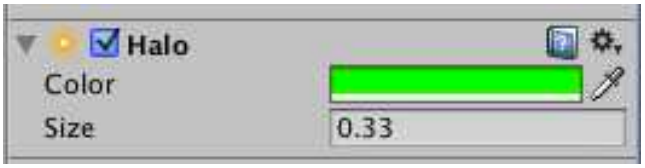
16. Sélectionnez **BalleRebond** dans la fenêtre *Hierarchy.*

16.1. Puis, à partir de la barre des menus, cliquez sur:

Component | Effets | Halo

17. Gardez la balle sélectionnée, puis :

17.1. Dans *l'Inspector,* changez:



Halo
17.1.1.***Color*** : Vert
17.1.2.***Size*** : 0.33

Maintenant, signalez au code que lorsqu'une collision est détectée avec un autre objet que la plate-forme, on remplace la balle par la fumée. On déplace ensuite la balle hors scène.

Ouvrez le script « gameRules.js ».

Ajoutez le code suivant :

```
…
var scoreGUI:UI.Text; //Champ de texte
var smoke:ParticleSystem;
…

function OnCollisionEnter(collision : Collision) {
    var colPos = collision.transform.position ;
    var rigid = GetComponent.<Rigidbody>();
    if (collision.collider.tag == "paddle"){
        var diffX = minTranslation+(colPos.x -
                        transform.position.x)*Random.Range(1,10);
        var diffZ = minTranslation+(colPos.z -
                        transform.position.z)*Random.Range(1,10);
        rigid.velocity = Vector3(-diffX, 8, -diffZ);
        addScore();
    }else{
        //GameOver
        smoke.transform.position = transform.position;
        transform.position = Vector3(-100,-100,-100);
        Time.timeScale = 1.0;
        smoke.Play();
    }
}
```

*Les textes en **gras** sont à ajouter, les **commentaires (//)** sont optionnels mais ils aident à comprendre le code, « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Lorsque la collision se fait avec (**paddle**), on donne 35 points. Lorsque celle-ci se fait avec quoi que ce soit d'autre, on place les particules à la Position de la balle. Ensuite, on replace la balle en dehors de la scène (**Vector3(-100, -100, -100)**), on réajuste la vitesse à 1.0 (**Time.timeScale = 1.0**) et on active les particules (**smoke.Play()**).

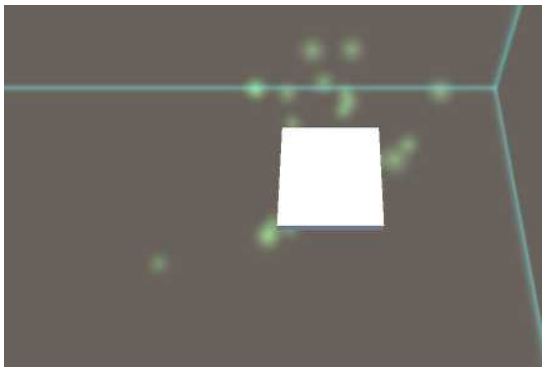
18. Sélectionnez **BalleRebond** dans la fenêtre *Hierarchy.*

18.1. Glissez le système de particules **Smoke** dans la nouvelle variable : ***smoke.***

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

Lorsque la balle touche un mur ou le sol, celle-ci part en fumée.



Retournez en mode édition.

Préparer le (build)

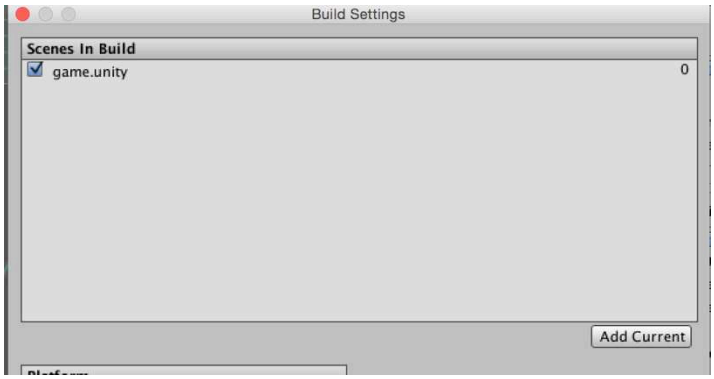
Bien joué! Évidemment, vous n'avez pas la possibilité de relancer le jeu lorsque vous perdez. C'est pourquoi, vous devez ajouter un bouton pour recharger la scène.

Vous devez d'abord et avant tout ajouter la scène actuelle dans le ***Build*** afin de pouvoir recharger celle-ci dans le code.

19. Dans la barre de menus, allez :

File | Build Settings

20. Appuyez sur le bouton « Add Current » pour ajouter la scène en cours, dans la liste.



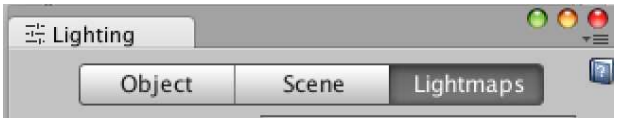
(Baker les lightmaps)

Maintenant, vous allez ***Baker*** les ***lightmaps*** de la scène pour que ceux-ci demeurent consistants lorsque vous rechargez la scène.

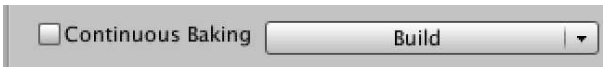
Par défaut, Unity met à jour ses **lightmaps** continuellement, mais il peut y avoir des problèmes quand une scène est rechargée dynamiquement. *(Du moins, c'est le cas dans la version 5.0).*

21. Dans la barre de menus, allez dans :

Window | Lighting



22. Dans le panneau *Lighting,* allez dans l'onglet **Lightmaps.**



22.1. Décochez ***Continuous Baking.***

22.2. Appuyez sur « Build » pour les calculer.

Vous êtes maintenant prêts à ajouter des boutons pour relancer ***lightmaps*** et quitter le jeu.

Créer les boutons de l'interface

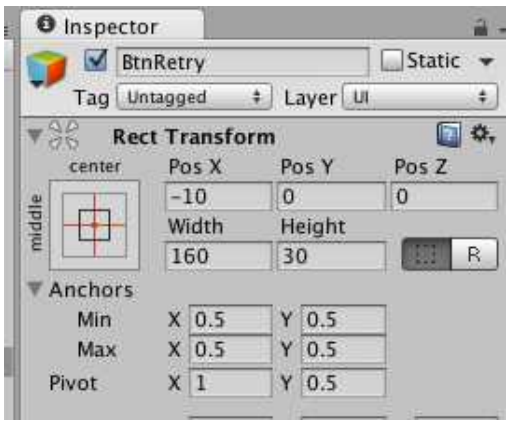
23. Sélectionnez le **canvas** dans votre *Hierarchy*.

24. À partir de la barre de menus :

GameObject | UI | Button

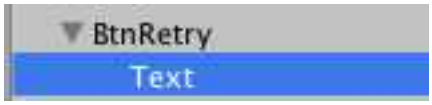
24.1. Nommez ce bouton : **BtnRetry.**

24.2. Dans *l'Inspector,* changez les paramètres suivants :



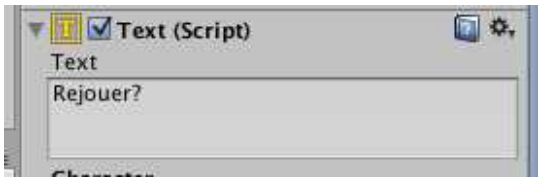
Rect Transform
24.2.1.***Pos X*= -10, *Pos Y*= 0, *Pos Z*= 0**
(Il est préférable de garder ces paramètres pour la fin)

24.2.2.***Width* : 160, *Height* : 30**
24.2.3.***Pivot* : X= 1, Y= 0.5**



25. Sélectionnez l'objet **Text** qui se trouve enfant du bouton **BtnRetry**.

25.1. Dans *l'Inspector,* changez la valeur du champ :



Text (Script)
25.1.1. ***Text*** : Rejouer?

26. Faites une copie du bouton **BtnRetry** en appuyant sur (**CTRL+D**) ou (**⌘+D**) sur Mac.

26.1. Nommez-le : **BtnQuit.**

26.2. Dans *l'Inspector,* changez les paramètres suivants :

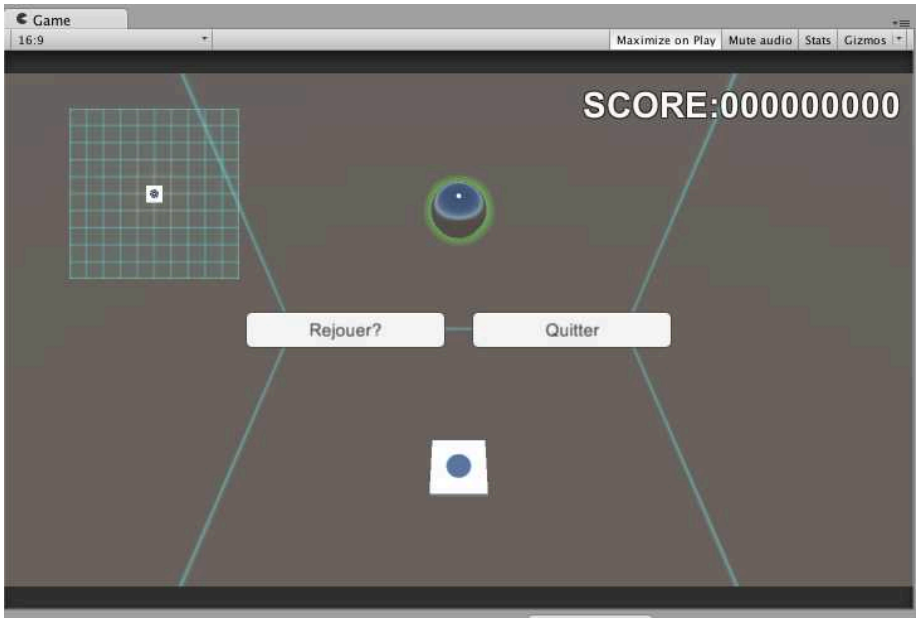
Rect Transform
26.2.1. ***Pos*** **X=** 10, ***Pos*** **Y=** 0, ***Pos*** **Z=** 0
26.2.2. ***Pivot*** **: X=** 0, **Y=** 0.5

27.Sélectionnez l'objet **Text** qui se trouve enfant du bouton **BtnQuit**.

27.1. Dans *l'Inspector,* changez la valeur du champ :

Text (Script)
27.1.1. ***Text*** : Quitter

Vous devriez obtenir un résultat similaire à celui-ci (les proportions peuvent changer).



Pour le code :

Ouvrez le script « gameRules.js ».

Ajoutez le code suivant :

```
…
var smoke:ParticleSystem;
var gameOver = false;
var btnQuit:GameObject;
var btnRetry:GameObject;
private var overOnce = false;

function Start () {
    gameOver = false;
    btnQuit.SetActive(false);
    btnRetry.SetActive(false);
}
…
function OnGUI() {
  scoreGUI.text = preTextScore + scoreTx;
  if (gameOver && !overOnce){
     overOnce = true;
     btnQuit.SetActive(true);
     btnRetry.SetActive(true);
  }
}

function quit(){
    Application.Quit ();
}

function restart(){
    Application.LoadLevel ("game");
}

function OnCollisionEnter(collision : Collision) {
    var colPos = collision.transform.position ;
    var rigid = GetComponent.<Rigidbody>();
    if (collision.collider.tag == "paddle"){
        var diffX = minTranslation+(colPos.x -
                            transform.position.x)*Random.Range(1,10);
        var diffZ = minTranslation+(colPos.z -
                            transform.position.z)*Random.Range(1,10);
        rigid.velocity = Vector3(-diffX, 8, -diffZ);
        addScore();
    }else{
        //GameOver
        smoke.transform.position = transform.position;
        transform.position = Vector3(-100,-100,-100);
        Time.timeScale = 1.0;
        smoke.Play();
        gameOver = true;
    }
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

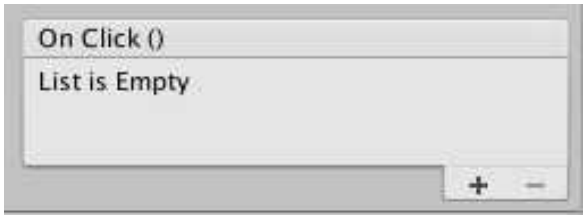
En détails...

Nous avons ajouté une variable pour marquer la fin du jeu (**gameOver**). Quand cette variable est vraie (**gameOver = true**), deux boutons sont affichés dans l'interface (**SetActive(true)**).

Si l'on appuie sur le bouton « Rejouer? », on charge alors la scène **Game** (**Application.LoadLevel (« game »)**), l'autre quitte le jeu (en version compilée).

Par contre, les boutons ne sont pas encore prêts pour ces actions.

28. Sélectionnez le bouton **BtnRetry** dans votre **canvas.**

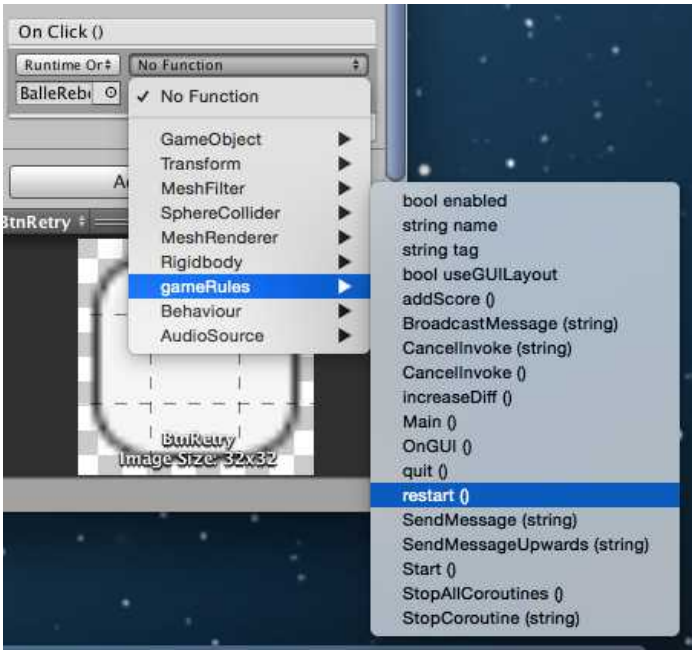


28.1. Dans *l'Inspector et* dans la section **On Click (),** appuyez sur le bouton **(+).**

28.2. Glissez **BalleRebond** dans le champ, en-dessous de ***Runtime Only.***

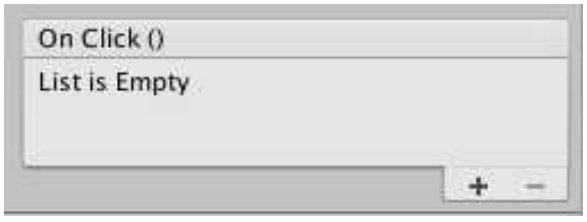
28.3. Dans le champ de droite, changez la valeur pour :

gameRules | restart ()

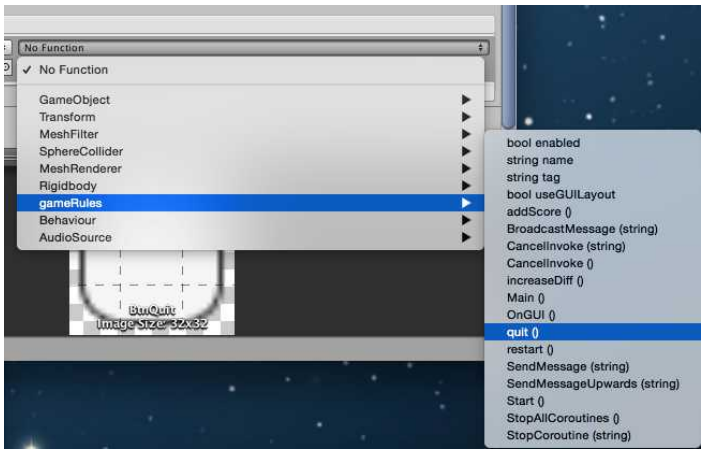


29. Sélectionnez le bouton **BtnQuit** dans votre **canvas.**

29.1. Dans *l'Inspector* et dans la section **On Click (),** appuyez sur le bouton **(+).**



29.2. Glissez **BalleRebond** dans le champ en-dessous de ***Runtime Only.***



29.3. Dans le champ « No Function », changez la valeur pour :

gameRules | quit ()

Les boutons ont maintenant le comportement d'assigner. Vous devez cacher les boutons, au début du jeu.

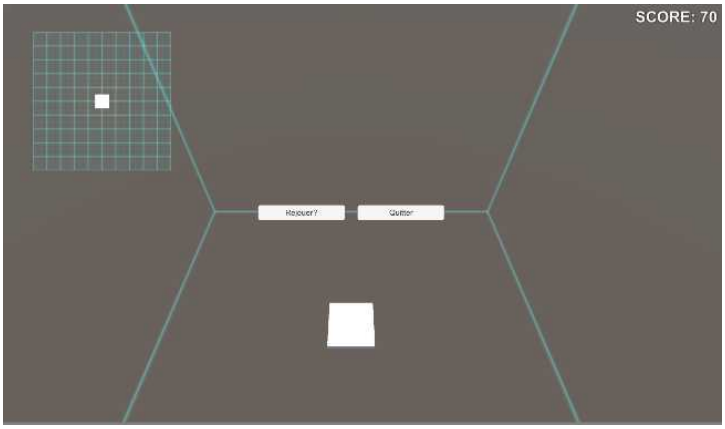
30. Sélectionnez l'objet : **BalleRebond.**

30.1 Glissez les boutons dans les champs :

30.1.1. ***Btn Quit :*** BtnQuit
30.1.2. ***Btn retry :*** BtnRetry

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.



Lorsqu'on perd, les boutons apparaissent à l'écran et nous permettent de relancer le niveau.

Retournez en mode édition.

Pour améliorer la jouabilité, nous allons modifier le script pour que le curseur de la souris soit invisible durant une partie, et visible quand le joueur perd ou lorsqu'il appuie sur la touche (**esc** ⎋) au clavier.

31. Ouvrez le script **movePaddle** et ajoutez :

```
…
function Start () {
 Cursor.lockState = CursorLockMode.Locked;
      Cursor.visible = false;
}

function Update () {
 if (Input.GetKey(KeyCode.Space)){
  Cursor.lockState = CursorLockMode.None;
  Cursor.visible = true;
 }
 var trans_Y : float = Input.GetAxis ("Mouse Y") * speed;
 var trans_X : float = Input.GetAxis ("Mouse X") * speed;
 trans_Y = trans_Y * (Time.deltaTime/Time.timeScale);
 trans_X = trans_X * (Time.deltaTime/Time.timeScale);
 transform.Translate (trans_X, 0, trans_Y);
}
…
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script

En détails…

Il faut figer et cacher le curseur de la souris au démarrage (**Start()**) de l'application. Si le joueur appuie sur la touche espace (**Input.GetKey(KeyCode.Space)**), on débloque alors le curseur.

Ouvrez le script « gameRules.js ».

Ajoutez le code suivant :

```
function OnCollisionEnter(collision : Collision) {
 var colPos = collision.transform.position ;
 var rigid = GetComponent.<Rigidbody>();
 if (collision.collider.tag == "paddle"){
  var diffX = minTranslation + (colPos.x -
                          transform.position.x)*Random.Range(1,10);
  var diffZ = minTranslation + (colPos.z -
                          transform.position.z)*Random.Range(1,10);
  rigid.velocity = Vector3(-diffX, 8, -diffZ);
  addScore();
 }else{
 //GameOver
  smoke.transform.position = transform.position;
  transform.position = Vector3(-100,-100,-100);
  Time.timeScale = 1.0;
  smoke.Play();
  gameOver = true;
  Cursor.lockState = CursorLockMode.None;
  Cursor.visible = true;
 }
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « ... » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Au moment où le joueur perd, on débloque également le curseur.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

Le curseur devrait rester au même endroit tout au long du jeu. Il sera invisible dans la version distribuable.

Retournez en mode édition.

L'habillage sonore

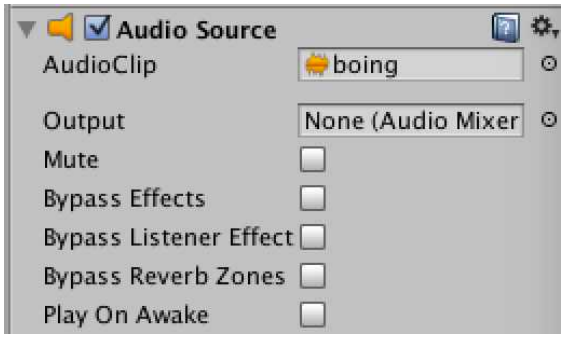
Maintenant que le *gameplay* de base est complet, vous devez ajouter des effets sonores.

32. Sélectionnez **Ballebond** dans le panneau *Hierarchy*.

32.1. Dans la barre des menus, allez dans :

Component | Audio | Audio Source

32.2. Dans *l'Inspector*, changez les paramètres suivants :



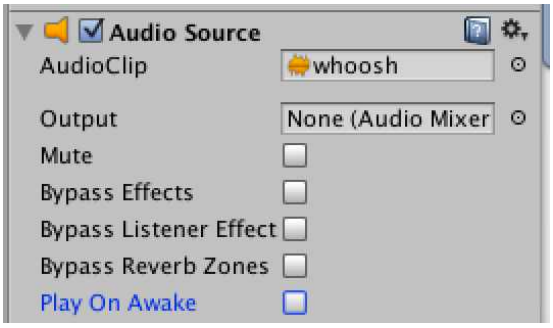
Audio Source
32.2.1. ***Audio Clip*** : (glissez le son **boing** qui se trouve dans *Project*).
32.2.2. ***Play On Awake*** : non

33. Sélectionnez **Smoke** dans *Hierarchy*.

33.1. Dans la barre des menus, allez dans :

Component | Audio | Audio Source

33.2. Dans *l'Inspector,* changez les paramètres suivants :



Audio Source
33.2.1. ***Audio Clip*** **:** (glissez le son **whoosh** qui se trouve dans *Project*).
33.2.2. ***Play On Awake*** **:** non

Ouvrez le script « gameRules.js ».

Ajoutez le code suivant :

```
…
function OnCollisionEnter(collision : Collision) {
    var colPos = collision.transform.position ;
    var rigid = GetComponent.<Rigidbody>();
    var audioPaddle = GetComponent.<AudioSource>();
    var audioSmoke = smoke.GetComponent.<AudioSource>();

    if (collision.collider.tag == "paddle"){
        audioPaddle.Play();
        var diffX = minTranslation + (colPos.x –
                            transform.position.x)*Random.Range(1,10);
        var diffZ = minTranslation + (colPos.z –
                            transform.position.z)*Random.Range(1,10);
        rigid.velocity = Vector3(-diffX, 8, -diffZ);
        addScore();
    }else{
        //GameOver
        smoke.transform.position = transform.position;
        transform.position = Vector3(-100,-100,-100);
        Time.timeScale = 1.0;
        smoke.Play();
        audioSmoke.Play();
        gameOver = true;
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

34. Dans votre *Hierarchy,* sélectionnez **Top Camera** et enlevez le ***Component*** : ***Audio Listener,*** car nous ne devons pas posséder 2 entrées de son 3D, en même temps.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

Votre jeu possède maintenant des effets sonores.

Retournez en mode édition.

En détails...

Ici, nous activons les effets sonores qui se trouvent sur la balle et sur l'émetteur de particules.

Voilà, le jeu est complet !

Conseil!

Sentez-vous libre de continuer le projet pour le rendre encore plus attrayant, visuellement!

Exporter un fichier redistribuable

Maintenant, vous allez exporter une version du jeu.

1. Sélectionnez l'image **icon** dans le panneau *Project,* puis :

1.1. Dans *l'Inspector,* changez :

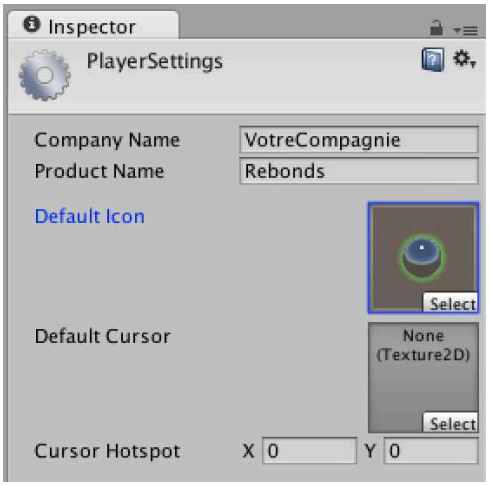
1.1.1. ***Texture Type*** : Sprite (2D and UI)
1.1.2. ***Sprite Mode*** : Single
1.1.3. ***Pixels Per Unit*** : 256
1.1.4. ***Max Size*** : 256
1.1.5. ***Format*** : 16 bits

1.2. Appuyez sur le bouton : « Apply ».

Voilà, l'icône est prête à être utilisée, l'image **splash** doit également être configurée de façon similaire.

Dans la barre de menus, allez dans :

Edit | Project Settings | Player



2.1. Dans *l'Inspector,* changez :

2.1.1. ***Company Name***: [Votre nom ou celui de votre entreprise]
2.1.2. ***Product Name*** : Rebonds
2.1.3. ***Default Icon*** : icon.png

Settings for PC, Mac & Linux Standalone

Resolution and Presentation
2.1.4. ***Default Is Full Screen***: oui
2.1.5. ***Default Is Native Resolution*** : oui

Standalone Player Options
2.1.6. ***Force Single Instance*** : oui

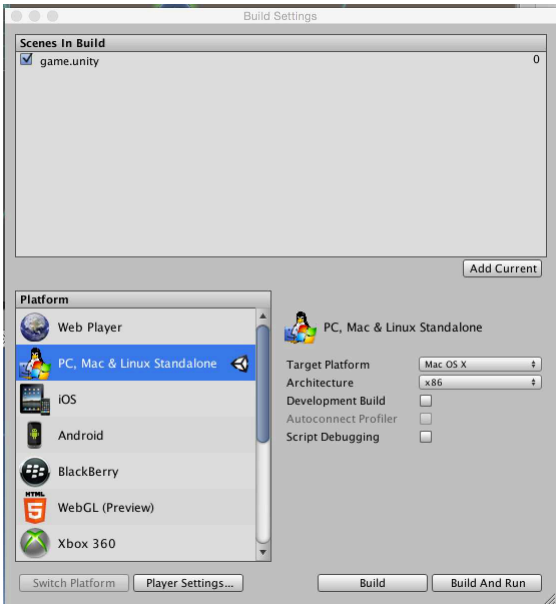
Laissez les autres paramètres de cette section tels quels.

Splash Image
2.1.7. ***Config Dialog Banner***: Splash

La taille de l'image **Splash** doit être : **432 x 163 pixels** et en mode ***Sprite (2D and UI).***

2. Dans la barre de menus, allez dans :

File | Build Settings…

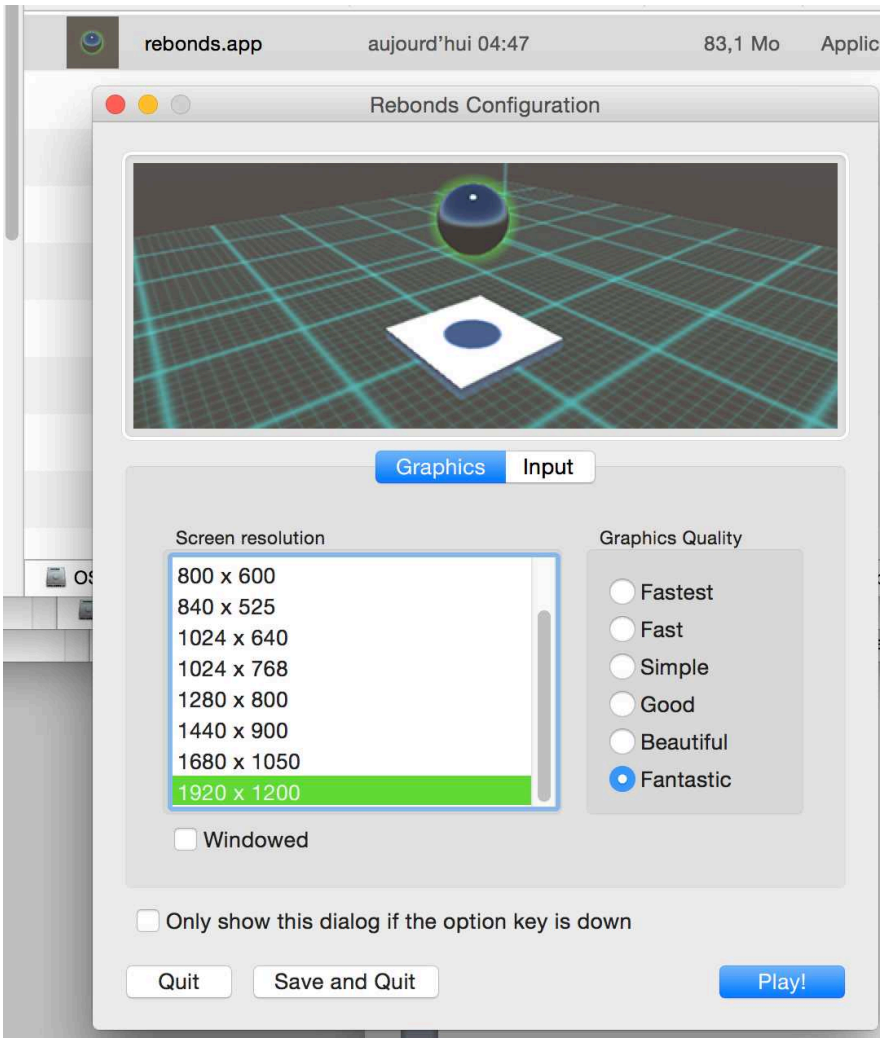


3. Dans le panneau ***Build Settings…,*** sélectionnez votre plateforme (Windows, Linux ou Mac Standalone), puis :

4.1. Appuyez sur le bouton « Build And Run ».

4.2. Sélectionnez un emplacement pour exporter votre jeu.

Le jeu sera compilé et se lancera automatiquement :



Voilà, l'exercice est complet et prêt à être distribué ! Si vous voulez recompiler une nouvelle version, vous n'avez qu'à utiliser le raccourci : (**CTRL+B**) ou (**⌘+B**) sur Mac.

File | Build And Run

Codes complétés

Voici les codes complets non-abrégés de l'exercice :

movePaddle

```
#pragma strict

var speed = 3.0;

function Start(){
  Cursor.lockState = CursorLockMode.Locked;
  Cursor.visible = false;
}

function Update () {
  if (Input.GetKey(KeyCode.Space)){
     Cursor.lockState = CursorLockMode.None;
     Cursor.visible = true;
  }
  var trans_Y : float = Input.GetAxis ("Mouse Y") * speed;
  var trans_X : float = Input.GetAxis ("Mouse X") * speed;
  trans_Y = trans_Y * (Time.deltaTime/Time.timeScale);
  trans_X = trans_X * (Time.deltaTime/Time.timeScale);
  transform.Translate (trans_X, 0, trans_Y);
}
```

gameRules (début)

```
#pragma strict

var minTranslation = 0.05;
var scoreboard = 0; //Pointage en chiffres.
var preTextScore = "SCORE: "; //Texte permanent.
var scoreTx = "0"; //Pointage en lettres.
var pointsPerBounce = 35; //Valeur d'un bond.
var scoreGUI:UI.Text; //Champ de texte
var smoke:ParticleSystem;
var gameOver = false;
var btnQuit:GameObject;
var btnRetry:GameObject;
private var overOnce = false;

function Start () {
  gameOver = false;
  btnQuit.SetActive(false);
  btnRetry.SetActive(false);
}

function addScore(){
  scoreboard += pointsPerBounce;
  scoreTx = scoreboard.ToString();
  increaseDiff();
}

function increaseDiff(){
  var timeScaleLimit:float = 3;
  var timeScaler:float = Time.timeScale + 0.03;
  if (timeScaler > timeScaleLimit)
     timeScaler = timeScaleLimit;
  Time.timeScale = timeScaler;
}

function OnGUI (){
  scoreGUI.text = preTextScore + scoreTx;
  if (gameOver == true && !overOnce){
     overOnce = true;
     btnQuit.SetActive(true);
     btnRetry.SetActive(true);
  }
}
```

gameRules (fin)

```
function OnCollisionEnter(collision : Collision) {
  var colPos = collision.transform.position ;
  var rigid = GetComponent.<Rigidbody>();
  var audioPaddle = GetComponent.<AudioSource>();
  var audioSmoke = smoke.GetComponent.<AudioSource>();
  if (collision.collider.tag == "paddle"){
    audioPaddle.Play();
    var diffX = minTranslation + (colPos.x -
                          transform.position.x)*Random.Range(1,10);
    var diffZ = minTranslation + (colPos.z -
                          transform.position.z)*Random.Range(1,10);
    rigid.velocity = Vector3(-diffX, 8, -diffZ);
    addScore();
  }else{
    //GameOver
    smoke.transform.position = transform.position;
    transform.position = Vector3(-100,-100,-100);
    Time.timeScale = 1.0;
    smoke.Play();
    audioSmoke.Play();
    gameOver = true;
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;
  }
}

function quit(){
   Application.Quit ();
}

function restart(){
   Application.LoadLevel ("game");
}
```

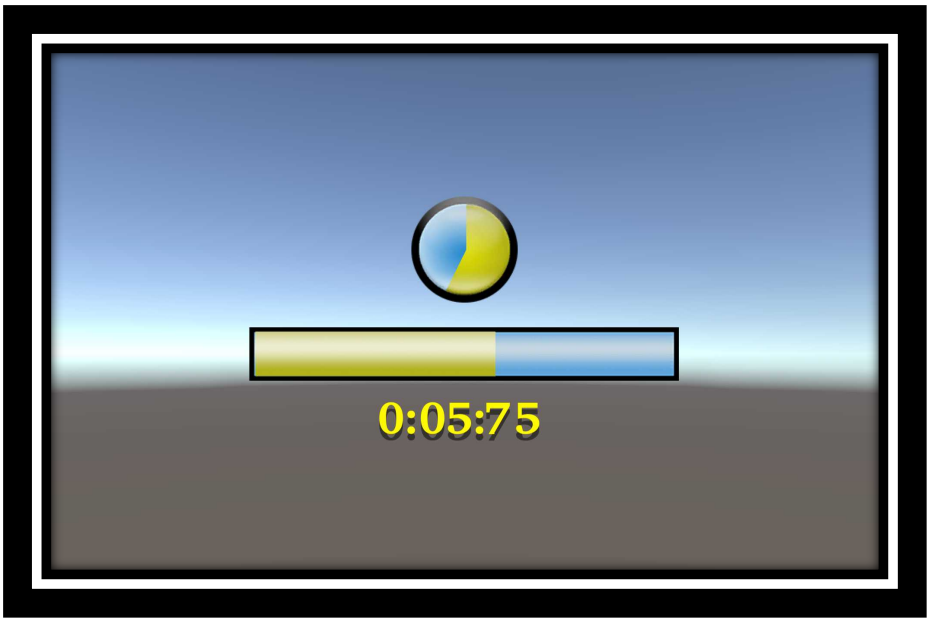
↘ *Certaines instructions ont été écrites sur plusieurs lignes à cause d'une contrainte d'espace.*

Chapitre 5 - Les chronomètres

Matière couverte dans ce chapitre

- **Création d'un chronomètre en chiffres**
- **Création d'un chronomètre en barre**
- **Création d'un chronomètre en pointe de tarte**

Maintenant que vous avez expérimenté avec le développement de jeu, vous allez découvrir quelque chose de différent mais qui est complémentaire à bien des projets, un chronomètre. Le temps est un facteur important dans les jeux vidéo depuis l'adoption de ceux-ci dans les salles d'arcades. Pour qu'un jeu soit rentable dans les arcades, un joueur doit mettre une pièce dans la machine, à toutes les 3 minutes.



Avant de débuter

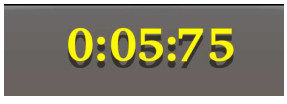
Vous aurez besoin du paquet : **Chapitre5.unitypackage** qui contient tous les éléments graphiques de l'exercice.

Atelier pratique

Préparatifs de la scène

Dans cet exercice, vous allez découvrir comment réaliser trois modèles de chronomètres pour vos jeux.

Une version texte



Une version en barre

Une version en pointe de tarte

Les éléments de la scène

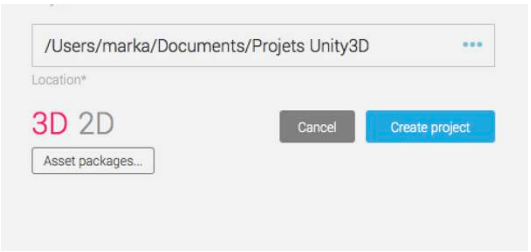
1. Ouvrez Unity®.

2. Cliquez sur le bouton : « New Project ».

3. Sauvez votre projet sur un disque dur local, en inscrivant le nom: **Chronos.**

3.1. Ce sera un projet : **3D.**

3.2. Appuyez sur « Create Project ».



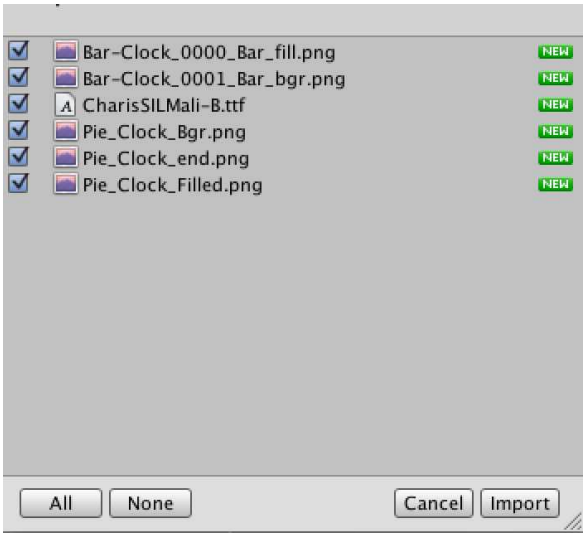
4. Lorsqu'Unity est ouvert, **sauvegardez votre scène** (**CTRL+S**) ou (**⌘+S**) sur Mac.

4.1. Nommez-la: **clocks**.

Avant de vous lancer dans le code, vous allez configurer vos éléments visuels dans la scène.

5. Avant de commencer, veuillez importer les fichiers de l'exercice dans le projet.

5.1. Dans le panneau *Project,* faites un clic secondaire, puis :



- **Import Package > Custom Package…**

5.2. Sélectionnez **Chapitre5.unitypackage** et importez son contenu.

Le paquet contient **5 images** pour les différents chronomètres et **une police**.

6. Sélectionnez les images.

6.1. Dans *l'Inspector,* changez leurs paramètres pour :

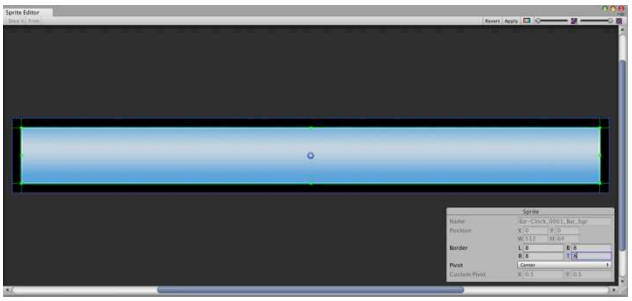
6.1.1. ***Texture Type*:** Sprite (2D and UI)
6.1.2. ***Sprite Mode***: Single
6.1.3. ***Pixels Per Unit*** : 100
6.1.4. ***Pivot*** : Center
6.1.5. ***Generate Mip Maps*** : Oui
6.1.6. ***Filter Mode*** : Bilinear
6.1.7. ***Max Size*** : 1024
6.1.8. ***Format*** : Truecolor

6.2. Appuyez sur le bouton « Apply ».

7. Une fois complété, désélectionnez toutes ***les images***, sauf pour **Bar_Clock_Bgr.**

7.1. Dans *l'Inspector,* appuyez sur le bouton : « Sprite Editor ».

7.2. Dans les paramètres du ***Sprite***, inscrivez les valeurs suivantes :



7.2.1. ***Border :*** **L=** 8**, R=** 8**, T=** 8**, B =** 8

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Ces modifications permettent à Unity de connaître quelles informations peuvent être déformées librement et lesquelles doivent rester uniformes. Dans cet exercice, la bordure noire devra rester uniforme.

8. Ensuite, à partir de la barre de menus, cliquez sur:

GameObject | UI | Canvas

9. Dans la section *Hierarchy*, sélectionnez le **Canvas.**

9.1. *Dans l'Inspector,* changez les valeurs de :

Canvas
9.1.1. ***Render Mode* :** Screen Space – Overlay
9.1.2. ***Pixel Perfect*** : oui
9.1.3. ***Sort Order*** : 0

Canvas Scaler
9.1.4. ***UI Scale Mode:*** Scale With Screen Size

9.1.5. ***Reference Resolution:*** **X**=1920, **Y**=1080
9.1.6. ***Screen Match Mode*** : Match Width Or Height
9.1.7. ***Match*** : Height = 1
9.1.8. ***Reference Pixels Per Unit*** : 100

Chrono en texte

Maintenant que vous avez un canevas, vous êtes prêts à ajouter un champ texte pour votre premier chronomètre.

10. Allez dans la barre de menus, puis faites :

GameObject | UI | Text

Un nouveau champ texte devrait apparaître, parfois, en dehors du cadre de l'écran. Il est donc possible que vous ne voyiez pas le texte dans la fenêtre *Game*.

11. Sélectionnez l'objet : **Text** dans votre *Hierarchy.*

11.1. *Dans l'Inspector,* changez les valeurs de:

Rect Transform
11.1.1. ***Pos*** **: X**=0, **Y**=-100, **Z**=0
(Il est préférable de garder ce paramètre pour la fin).

11.1.2. ***Width***=264, ***Height***= 108
11.1.3. ***Anchors Min*** **X**=0.5, ***Min*** **Y**=0.5
11.1.4. ***Anchors Max*** **X**=0.5, ***Max*** **Y**=0.5
11.1.5. ***Pivot*** **X**=0.5, **Y**=0.5

Text (Script)
11.1.6. ***Text*** **:** 0 :10 :00

Character
11.1.7. ***Font*** **:** CharisSILMali-B
11.1.8. ***Font Style*** **:** Normal
11.1.9. ***Font Size*** **:** 70
11.1.10. ***Line Spacing*** **:** 1
11.1.11. ***Rich Text*** **:** Oui

Paragraph
11.1.12. ***Alignment*** **:** Gauche, Millieu

11.1.13. ***Horizontal Overflow*:** Overflow
11.1.14. ***Vertical Overflow*:** Overflow
11.1.15. ***Best Fit* :** non
11.1.16. ***Color* :** Jaune
11.1.17. ***Material* :** aucun

Par défaut, le texte est en aplat à l'écran. Vous allez ajouter un effet d'ombre pour détacher le texte de l'arrière-plan.

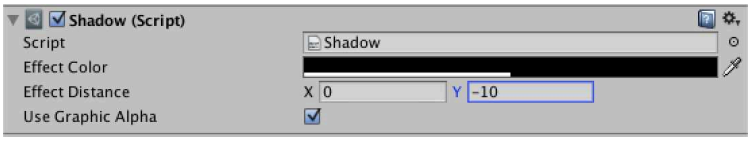
12. Dans *l'Inspector*, cliquez sur le bouton : « Add Component » puis :

UI | Effect | Shadow

Une nouvelle boîte de paramètres va s'ajouter à la liste :

12.1. Changez les paramètres dans votre *Inspector*:

Shadow (Script)
12.1.1. ***Effect Color*:** Noir (semi-transparent)
12.1.2. ***Effect Distance:*** **X**=0, **Y**=-10
12.1.3. ***Use Graphic Alpha*:** oui



Vous allez maintenant préparer le script pour faire fonctionner le tout.

JavaScript

13. Dans le panneau *Project,* faites un clic secondaire puis :

- **Create > Javascript**

13.1. Nommez-le : **clockScript.**

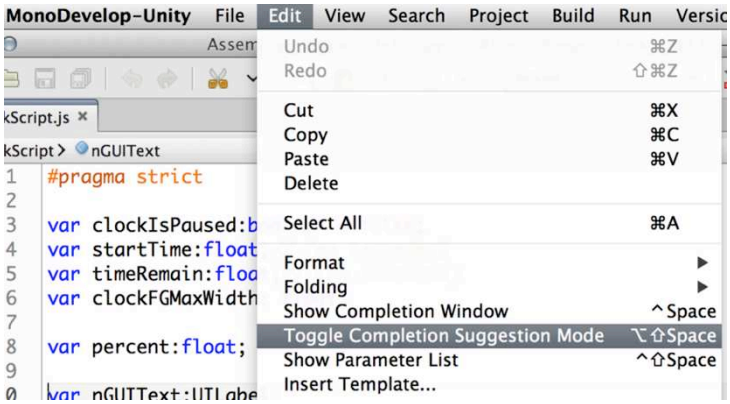
14. Glissez votre script sur l'objet **EventSystem** dans le panneau *Hierarchy*. Maintenant que votre script est attaché à la scène, vous pouvez le modifier.

Ouvrez ce fichier (avec *MonoDevelop* ou un autre éditeur de script).

Si vous utilisez *MonoDevelop*, avant de commencer, pensez à désactiver l'auto-remplissage des commandes (***Toggle Completion Suggestion Mode***) car celui-ci nuit plus souvent qu'il aide.

• À partir de la barre de menus de *Mono-Develop,* appuyez sur :

Edit | Toggle Completion Suggestion Mode



Maintenant, ajoutez ce bloc de codes :

```
var Chrono_duree:float = 10.0;// (en secondes)
private var timeRemain:float;// (en secondes)
var clockIsPaused:boolean = false;

function Start() {

}

function Update () {
 if (clockIsPaused == false){
  DoCountDown();
 }
}

function SetDuration(){

}

function ShowTime(){

}

function DoCountDown() {

}

function TimeIsUp(){

}
```

En détails…

Présentement, vous avez préparé votre structure avec les fonctions nécessaires même si celles-ci sont vides. Vous avez une variable (**clockIsPaused**) que vous avez initialisée à **faux** afin de pouvoir arrêter ce chrono, quand le temps arrive à 0. Il y a aussi les fonctions : (**DoCountDown**()) pour faire le compte à rebours, (**startClock()**) afin de pouvoir activer ou désactiver (**clockIsPaused**), (**ShowTime()**), pour rafraîchir le visuel et (**TimeIsUp()**) à la toute fin, lorsque le compteur atteint **0:00:00**.

Pour la logique :

Ouvrez le script « clockScript.js »

Ajoutez ce bloc de codes :

```
var Chrono_duree:float = 10.0;// (en secondes)
private var timeRemain:float;// (en secondes)
var clockIsPaused:boolean = false;

function Start() {
    SetDuration();
}
…
function SetDuration(){
    timeRemain = Chrono_duree;
}
…
function DoCountDown() {
   timeRemain = Chrono_duree - Time.time;
   if (timeRemain < 0){
       timeRemain = 0;
       TimeIsUp();
   }
}
…
function TimeIsUp(){
   Debug.Log("Time Out!");
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

Après 10 secondes, vous devriez voir un message s'afficher dans la console *Debug*.

Retournez en mode édition.

Désormais, vous savez que la logique fonctionne. Maintenant, vous allez faire l'affichage à l'écran.

Ouvrez le script « clockScript.js »

Ajoutez ce bloc de codes :

```
var Chrono_duree:float = 10.0;// (en secondes)
private var timeRemain:float;// (en secondes)
var clockIsPaused:boolean = false;
var GUIText:UI.Text;
…

function ShowTime(){
  var minutes:int;
  var secondes:int;

  var timeStr:String;
  minutes = timeRemain/60;
  secondes = timeRemain % 60; // % = modulo

  timeStr = minutes.ToString() + ":" + secondes.ToString("D2");
  // (« D2 ») On precise que l'on veut absolument 2 decimales.
  GUIText.text = timeStr;
}

function DoCountDown() {
    timeRemain = Chrono_duree - Time.time;
    if (timeRemain < 0){
        timeRemain = 0;
        TimeIsUp();
    }else{
        ShowTime();
    }
}
…
```

*Les textes en **gras** sont à ajouter, les **commentaires (//)** sont optionnels mais ils aident à comprendre le code et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

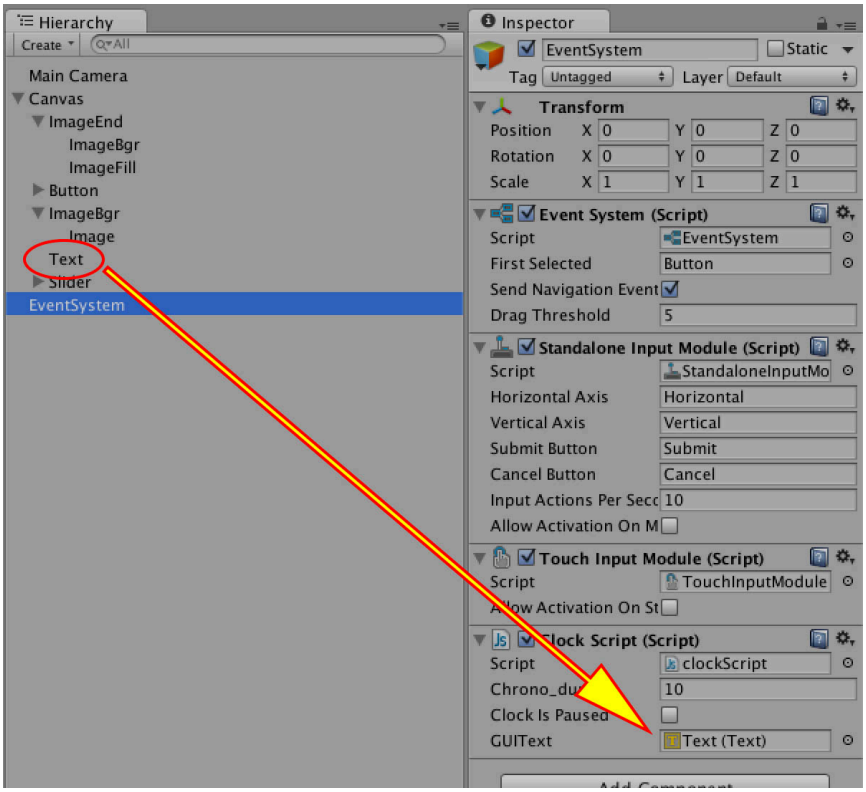
Dans les lignes de codes ajoutées, il y a le symbole (**%**) pour modulo. Le résultat du modulo est le reste d'une division (dans le cas présent, 60). Également, (**ToString(« D2 »)**) le (**D2**) signifie qu'il faut absolument garder 2 décimales visibles en tout temps. Donc, il faut voir 2 : 20 et non 2 : 2.

Maintenant, il faut spécifier notre champ **Text** comme valeur pour la variable : ***GUIText***.

15. Sélectionnez **EventSystem** dans le panneau *Hierarchy*,

15.1. Dans *l'Inspector,* glissez :

15.1.1. **GUI Text** : glissez l'objet **Text** dans la variable.



Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

Le compteur devrait s'afficher.

Voilà pour le chronomètre en texte, mais la dernière seconde ne se voit pas, car les millisecondes ne s'affichent pas.

Retournez en mode édition.

Ouvrez le script « clockScript.js »

Ajoutez les lignes suivantes :

```
…
function ShowTime(){
   var minutes:int;
   var secondes:int;
   var millisec:int;
   var timeRemainRounded:int = timeRemain;
   var timeStr:String;
   minutes = timeRemain/60;
   secondes = timeRemain % 60; // % = modulo
   millisec = (timeRemain-timeRemainRounded)*100;
   timeStr = minutes.ToString() + ":" + secondes.ToString("D2") +
                                        ":"+ millisec.ToString("D2");
   // (« D2 ») On precise que l'on veut absolument 2 decimales.
   GUIText.text = timeStr;
}
…
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails…

D'abord, il faut placer un nombre réel [ex. :7.1733] (**timeRemain**) dans un nombre entier [ex. :7] (**timeRemainRounded**), ce qui supprime les décimales. Ensuite, nous calculons la différence afin de garder uniquement les décimales [ex. :7.1733 – 7 = 0.1733] et nous multiplions ce chiffre par 100 pour obtenir un nombre entier [ex. : 0.1733 * 100 = 17.33 -> **17**]. Finalement, nous affichons ce nombre à la fin du texte (**timeStr**).

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

Retournez en mode édition.

Vous allez maintenant créer différentes interfaces pour rendre le chrono plus attrayant.

Chrono en barre

16. Dans la section *Hierarchy*, sélectionnez **Canvas.**

16.1. Allez dans la barre de menus :

GameObject | UI | Image

Un carré blanc devrait apparaître.

17. Dans la section *Hierarchy*, sélectionnez l'image.

17.1. Nommez l'image : **ImageBgr.**

17.2. changez les paramètres suivants :

Rect Transform
17.2.1. ***Pos X*** = 0, ***Y*** = 0, ***Z*** = 0
(Il est préférable de garder ce paramètre pour la fin).

17.2.2. ***Width*** **:** 655.36, ***Height*** : 81.92
17.2.3. ***Anchors Min*** **X**=0.5, ***Min*** **Y**=0.5
17.2.4. ***Anchors Max*** **X**=0.5, ***Max*** **Y**=0.5
17.2.5. ***Pivot*** **X**=0.5, **Y**=0.5

Image (Script)
17.2.6. ***Source Image*****:** Bar_Clock_Bgr
17.2.7. ***Color*****:** Blanc
17.2.8. ***Material*****:** aucun
17.2.9. ***Image Type*****:** Sliced
17.2.10. ***Fill Center*****:** oui

18. Dans la section *Hierarchy*, sélectionnez **Canvas.**

18.1. Allez dans la barre de menus, puis :

GameObject | UI | Image

Un nouveau carré blanc devrait apparaître.

19. Mettez l'image que vous venez de créer, enfant d'**ImageBgr.**

19.1. Nommez-la : **ImageFilled.**

19.2. Changez les paramètres suivants :

Rect Transform
19.2.1. ***Pos X*** = 0***, Y*** = 0***, Z*** = 0
(Il est préférable de garder ce paramètre pour la fin).

19.2.2. ***Width*** **:** 637, ***Height*** : 68
19.2.3. ***Anchors :*** **Min X**=0.5, **Min Y**=0.5
19.2.4. ***Anchors :*** **Max X**=0.5, **Max Y**=0.5
19.2.5. ***Pivot*** **: X**=0.5, **Y**=0.5

Image (Script)
19.2.6. ***Source Image*****:** Bar_Clock_Filler
19.2.7. ***Color*****:** Blanc
19.2.8. ***Material*****:** aucun
19.2.9. ***Image Type*****:** Filled
19.2.10. ***Fill Method*****:** Horizontal
19.2.11. ***Fill Origin*** **:** Left
19.2.12. ***Fill Amount*** **:** 1
19.2.13. ***Préserve Aspect*** **:** non

La partie visuelle est maintenant terminée.

Ouvrez le script « clockScript.js »

Ajoutez les lignes suivantes :

```
var Chrono_duree:float = 10.0;// (en secondes)
private var timeRemain:float;// (en secondes)
var clockIsPaused:boolean = false;
var GUIText:UI.Text;
var percent:float;
var clockFG:UI.Image;
…
function Update () {
  if (clockIsPaused == false){
      DoCountDown();
      clockFG.fillAmount = percent;
  }
}
…
function DoCountDown() {
      timeRemain = Chrono_duree - Time.time;
      percent = timeRemain / Chrono_duree;
      if (timeRemain < 0){
           timeRemain = 0;
           TimeIsUp();
      }else{
           ShowTime();
      }
}
…
function TimeIsUp(){
     Debug.Log("Time Out!");
     GUIText.alignment = TextAnchor.MiddleCenter;
     GUIText.text = "Time out";
     clockIsPaused = true;
}
```

↘ *Les textes en **gras** sont à ajouter ou ~~**retirer,**~~ le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails…

Les changements calculent le pourcentage [entre 0 et 1] de temps restant, avec les propriétés d'Unity® 5. Nous utilisons (**fillAmount**) pour remplir l'image. De plus, quand le temps est écoulé, nous changeons le texte pour (**Time out**), aligné au centre.

20. Sélectionnez **EventSystem** dans *Hierarchy.*

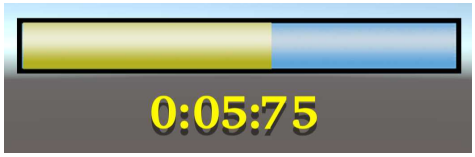
20.1. Dans *l'Inspector*, changez la valeur de :

20.1.1. ***Clock FG :*** Glissez l'objet **ImageFilled** de votre *Hierarchy.*

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.

Tout devrait fonctionner, il ne reste plus qu'à faire de la place pour le dernier chrono.



Retournez en mode édition.

Le chrono en pointe de tarte

Avant l'arrivée du système d'Unity® 4.6, faire ce type de chrono demandait plusieurs couches superposées afin de créer l'effet de progression. Maintenant, réaliser cet effet est aussi simple que faire le chrono en barre.

21. Dans la section *Hierarchy*, sélectionnez le **Canvas.**

21.1. À partir de la barre de menus, allez dans :

GameObject | UI | Image

22. Nommez votre image : **PieImgEnd.**

22.1. Changez les paramètres :

Rect Transform
22.1.1. ***Pos*** **X** = 0**, Y** = 160**, Z** = 0
(Il est préférable de garder ce paramètre pour la fin).

22.1.2. ***Width*** **:** 164, ***Height*** : 164
22.1.3. ***Anchors :*** **Min X**=0.5, **Min Y**=0.5
22.1.4. ***Anchors :*** **Max X**=0.5, **Max Y**=0.5
22.1.5. ***Pivot*** **: X**=0.5, **Y**=0.5

Image (Script)
22.1.6. ***Source Image :*** Pie_Clock_end
22.1.7. ***Color :*** Blanc
22.1.8. ***Material :*** Aucun

22.1.9. ***Image Type :*** Simple
22.1.10. ***Preserve Aspect :*** oui

Vous devriez avoir un fond rouge pour la fin du chrono. Vous allez ajouter un fond bleu pour le chrono actif.

23. Dans la section *Hierarchy*, sélectionnez **Canvas.**

23.1. À partir de la barre de menus, allez à :

GameObject | UI | Image

23.2. Nommez votre image : **PieImgBgr.**

23.3. Mettez **PieImgBgr** enfant de **PieImgEnd** et changez les paramètres de la nouvelle image dans *l'Inspector* :

Rect Transform
23.3.1. ***Pos X*** = 0, ***Y*** = 0, ***Z*** = 0
(Il est préférable de garder ce paramètre pour la fin)

23.3.2. ***Width*** **:** 164, ***Height*** : 164
23.3.3. ***Anchors :*** **Min X**=0.5, **Min Y**=0.5
23.3.4. ***Anchors :*** **Max X**=0.5, **Max Y**=0.5
23.3.5. ***Pivot*** **: X**=0.5, **Y**=0.5

Image (Script)
23.3.6. ***Source Image*** **:** Pie_Clock_Bgr
23.3.7. ***Color*** **:** Blanc
23.3.8. ***Material*** **:** aucun
23.3.9. ***Image Type*** **:** Simple
23.3.10. ***Preserve Aspect*** **:** oui

Vous avez maintenant un cercle bleu comme arrière-plan.

24. Dans la section *Hierarchy*, sélectionnez le **Canvas.**

24.1. Puis, dans la barre de menus :

GameObject | UI | Image

24.2. Nommez votre image : **PieImgFill**.

24.3. Mettez **PieImgFill** enfant de **PieImgBgr.**

25. Sélectionnez le **PieImgFill.**

25.1. Changez les paramètres suivants dans *l'Inspector* :

Rect Transform
25.1.1. ***Pos X*** = 0, ***Y*** = 0, ***Z*** = 0
(Il est préférable de garder ce paramètre pour la fin).

25.1.2. ***Width*** **:** 164, ***Height*** : 164
25.1.3. ***Anchors :*** **Min X**=0.5, **Min Y**=0.5
25.1.4. ***Anchors :*** **Max X**=0.5, **Max Y**=0.5
25.1.5. ***Pivot*** **: X**=0.5, **Y**=0.5

Image (Script)
25.1.6. ***Source Image*** **:** Pie_Clock_Filled
25.1.7. ***Color*** **:** Blanc
25.1.8. ***Material*** **:** aucun
25.1.9. ***Image Type*** **:** Filled
25.1.10. ***Fill Method*** **:** Radial 360
25.1.11. ***Fill Origin*** **:** Top
25.1.12. ***Fill Amount*** **:** 1
25.1.13. ***Clockwise*** **:** oui
25.1.14. ***Preserve Aspect*** **:** oui

Si vous changez manuellement la valeur de **Fill Amount** dans *l'Inspector*, vous verrez comment fonctionne le remplissage :

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Ouvrez le script « clockScript.js ».

Ajoutez les lignes suivantes :

```
…
var clockFG:UI.Image;
var Pie_fill:UI.Image;
var Pie_bgr:UI.Image;
var Pie_fin:UI.Image;
…
function Update () {
  if (clockIsPaused == false){
     DoCountDown();
     clockFG.fillAmount = percent;
     Pie_fill.fillAmount = percent;
  }
}
…
function SetDuration(){
   timeRemain = Chrono_duree;
   clockIsPaused = false;
   Pie_bgr.enabled = true;
   Pie_fill.enabled = true;
   Pie_fin.enabled = false;
}
…
function TimeIsUp(){
   GUIText.alignment = TextAnchor.MiddleCenter;
   GUIText.text = "Time out";
   clockIsPaused = true;
   Pie_bgr.enabled = false;
   Pie_fin.enabled = true;
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

En détails…

Il faut utiliser (**fillAmount**) pour remplir l'image de la même façon que le chrono en barre. Également, quand le temps est écoulé, nous cachons l'image vide pour l'image complétée (**Pie_fin.enabled = true**).

Sauvegardez le script et retournez dans l'éditeur.

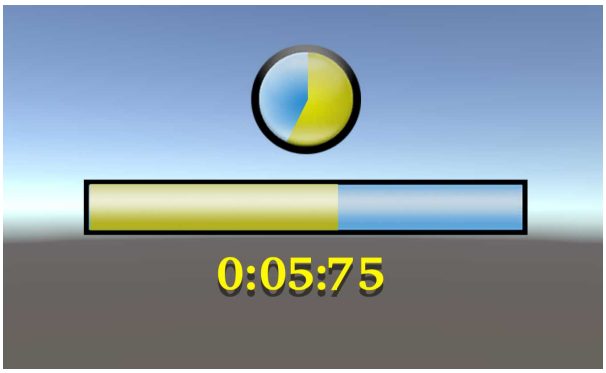
26. Cliquez sur **EventSystem** dans *Hierarchy.*

26.1. Dans l'Inspector, changez les paramètres suivants dans le script **clockScript** :

26.1.1. ***Pie_fill :*** Glissez (PieImgFill)
26.1.2. ***Pie_bgr:*** Glissez (PieImgBgr)
26.1.3. ***Pie_fill :*** Glissez (PieImgFill)

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez votre jeu.



Voilà, vous avez complété cet exercice. Sentez-vous libre d'incorporer un de ces chronomètres dans vos productions futures, au besoin.

Retournez en mode édition.

Codes complétés

Voici le code complet non-abrégé de l'exercice :

clockScript

```
#pragma strict

var Chrono_duree:float = 10.0; // (en secondes)
private var timeRemain:float; // (en secondes)
var clockIsPaused:boolean = false;
var GUIText:UI.Text;
var percent:float;
var clockFG:UI.Image;
var Pie_fill:UI.Image;
var Pie_bgr:UI.Image;
var Pie_fin:UI.Image;


function Start() {
    SetDuration();
}

function Update () {
    if (clockIsPaused == false){
        DoCountDown();
        clockFG.fillAmount = percent;
        Pie_fill.fillAmount = percent;
    }
}

function SetDuration(){
    timeRemain = Chrono_duree;
    clockIsPaused = false;
    Pie_bgr.enabled = true;
    Pie_fill.enabled = true;
    Pie_fin.enabled = false;
}
```

clockScript (fin)

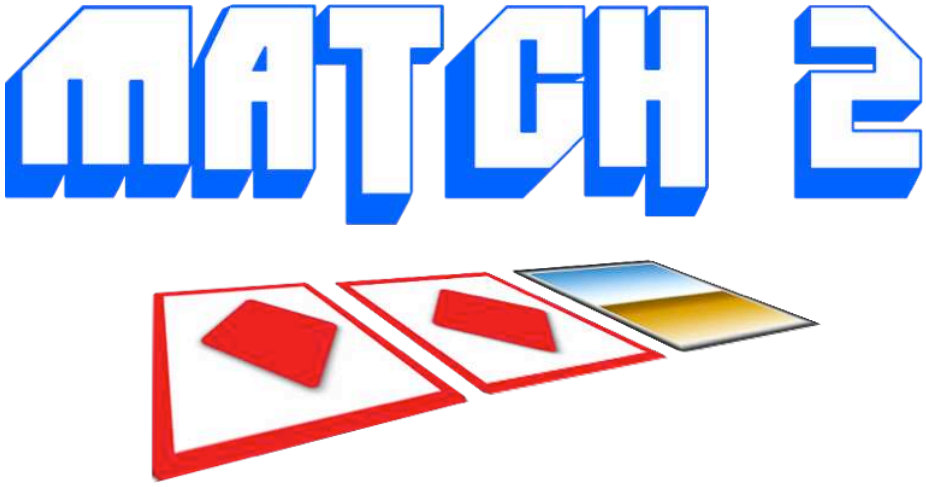
```
function ShowTime(){
    var minutes:int;
    var secondes:int;
    var millisec:int;
    var timeRemainRounded:int = timeRemain;
    var timeStr:String;
    minutes = timeRemain/60;
    secondes = timeRemain % 60; // % = modulo
    millisec = (timeRemain-timeRemainRounded)*100;
    timeStr = minutes.ToString() + ":" + secondes.ToString("D2")+
                                        ":" + millisec.ToString("D2");
    // (« D2 ») On precise que l'on veut absolument 2 decimales.
    GUIText.text = timeStr;
}

function DoCountDown() {
    timeRemain = Chrono_duree - Time.time;
    percent = timeRemain / Chrono_duree;
    if (timeRemain < 0){
        timeRemain = 0;
        TimeIsUp();
    }else{
        ShowTime();
    }
}

function TimeIsUp(){
    GUIText.alignment = TextAnchor.MiddleCenter;
    GUIText.text = "Time out";
    clockIsPaused = true;
    Pie_bgr.enabled = false;
    Pie_fin.enabled = true;
}
```

unity

Chapitre 6 - Jeu *#2 : Match-deux*

Matière couverte dans ce chapitre

- **Création d'un jeu en 2D**
- **Assemblage d'un menu principal**
- **Utilisation des interfaces utilisateurs (UI)**
- **Création d'une scène de jeu**
- **Programmation du jeu.**



Vous allez réaliser un petit jeu de mémoire intitulé : **Match 2.**

Ce jeu de cartes classique consiste à retrouver toutes les paires de cartes, en retournant deux cartes à la fois.

Si vous découvrez une paire, vous laissez les cartes dévoilées au joueur et vous continuez à jumeler les autres.

Si au contraire, vous découvrez deux cartes différentes, vous retournez celles-ci. Dans certains cas, le joueur à un certain nombre d'essais pour jumeler toutes les cartes. Dans d'autres cas, on pénalise le joueur quand il tente infructueusement de faire des associations, plus d'une fois.

Avant de débuter

Vous aurez besoin du paquet : **Chapitre6.unitypackage** qui contient tous les éléments graphiques de l'exercice.

Atelier pratique

Préparation du menu principal

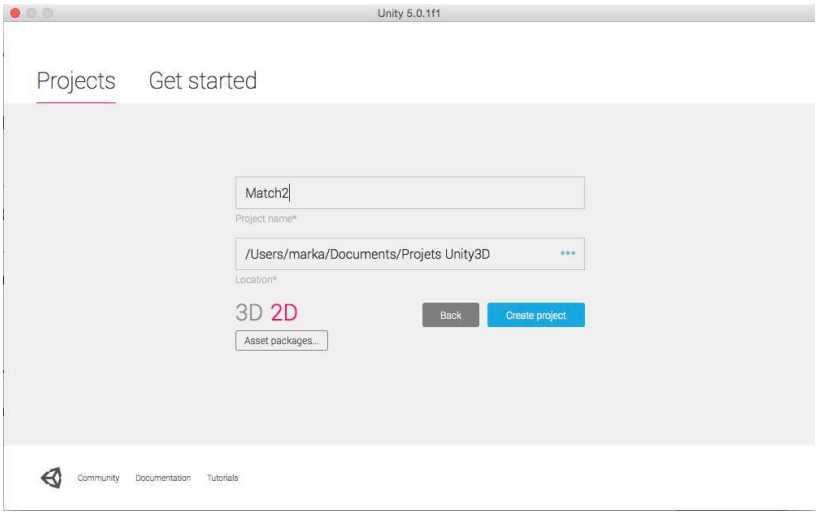
Dans cet exercice, nous introduirons ces nouveaux éléments : un menu principal au démarrage du jeu et l'utilisation de plusieurs scènes.

Les éléments de la scène

1. Ouvrez Unity®.

2. Cliquez sur le bouton : « New Project ».

3. Sauvez votre projet sur un disque dur local en inscrivant le nom : **Match2.**



3.1. Ce sera un projet : **2D.**

3.2. Appuyez sur « Create Project ».

4. Lorsque le projet est ouvert, **sauvegardez votre scène** (**CTRL+S**) ou (**⌘+S**) sur Mac.

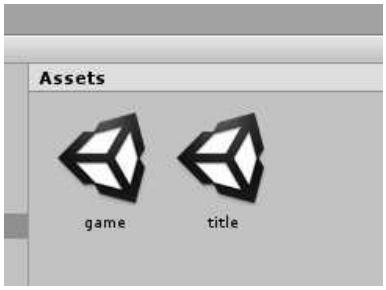
4.1. Nommez la scène : **game.**

5. Créez une deuxième scène en appuyant sur (**CTRL+N**) ou (**⌘+N**) sur Mac**.**

6. **Sauvegardez la scène** (**CTRL+S**) ou (**⌘+S**) sur Mac.

6.1. Nommez la 2e scène : **title.**

7. Ouvrez la scène **title** en cliquant dessus, dans le panneau *Project.*



Avant d'aller plus loin dans le code, vous allez configurer vos éléments visuels dans la scène.

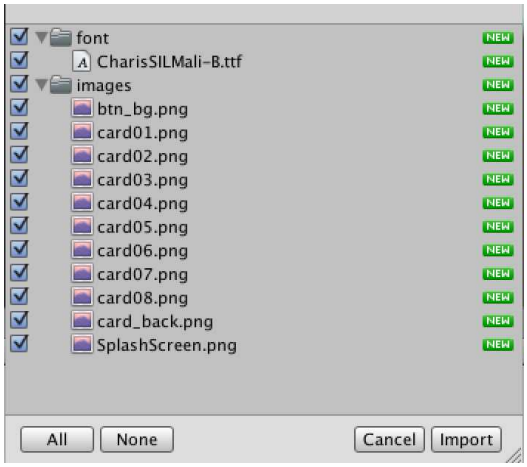
8. Importez les fichiers de l'exercice dans le projet.

8.1. Dans un espace vide du panneau *Project*, faites un clic secondaire, puis :

- **Import Package > Custom Package…**

8.2. Sélectionnez **Chapitre6.unitypackage** et importez son contenu.

Vous devriez avoir 2 nouveaux répertoires avec leur contenu :

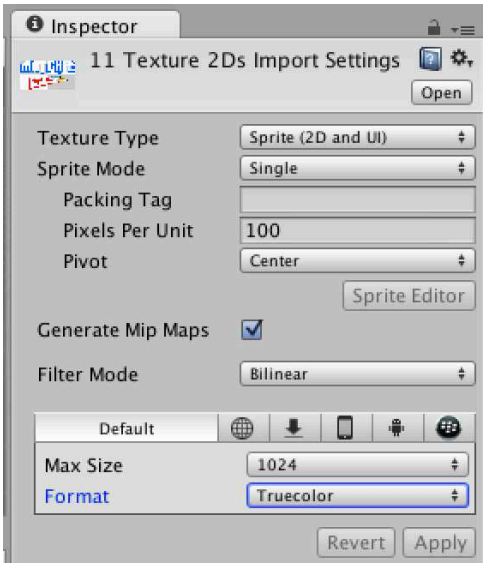


9. Dans le panneau *Project*, ouvrez le répertoire : **images.**

10. Sélectionnez toutes les images de ce répertoire.

10.1. Dans *l'Inspector*, changez les valeurs :

10.1.1. ***Texture Type*** : Sprite (2D and UI)
10.1.2. ***Sprite Mode*** : Single
10.1.3. ***Pivot*** : Center
10.1.4. ***Generate Mip Maps*** : Oui
10.1.5. ***Filter Mode*** : Bilinear
10.1.6. ***Format*** : Truecolor



10.2. Appuyez sur le bouton : « Apply ».

Ces paramètres permettent d'avoir des textures qui ne sont pas en puissance de deux et qui gardent toutes leurs qualités visuelles et alphas. Ils sont également essentiels pour les éléments des interfaces (**UI**).

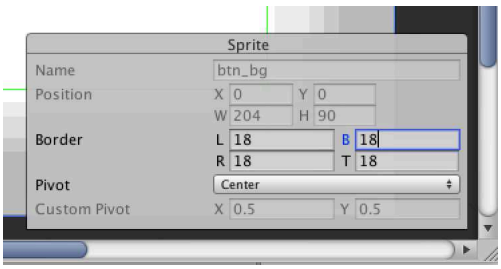
Avant de débuter la création de votre scène, vous allez configurer les paramètres de l'objet : **btn_bg** afin de définir des zones qui ne doivent pas s'étirer, vous permettant de réajuster la taille d'un bouton, sans le déformer.

11. Dans le panneau *Project*, sélectionnez l'image : **btn_bg.**

12. Dans *l'Inspector,* appuyez sur le bouton « Sprite Editor ».

12.1. Dans le panneau *Sprite,* changez les paramètres suivants :

12.1.1. ***Border*** : **L=** 18, **R=** 18, **B=** 18, **T=** 18



13. Fermez la fenêtre.

14. Sauvegardez les changements avec le bouton « Apply ».

Le bouton est maintenant prêt, vous pouvez construire votre interface.

15. Dans la barre de menus, allez à :

GameObject | UI | Canvas

16. Vous devriez maintenant avoir trois éléments dans *Hierarchy* :

- **Main Camera**
- **Canvas**
- **EventSystem**

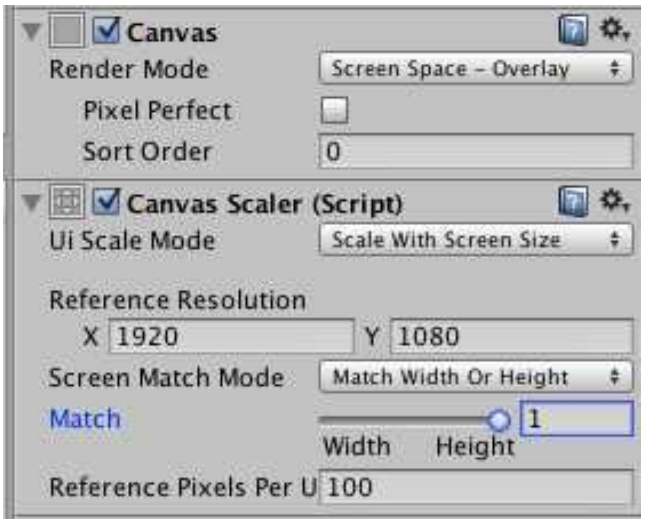
17. Sélectionnez : **Canvas**

17.1. Nommez-le : **CanvasMenu**

17.2. Changez les paramètres suivants :

Canvas
17.2.1. ***Render Mode*** : Screen Space – Overlay
17.2.2. ***Pixel Perfect*** : non
17.2.3. ***Sort Order*** : 0

Canvas Scaler (Script)
17.2.4. ***UI Scale Mode*** : Scale With Screen Size
17.2.5. ***Reference Resolution*** **: X=** 1920**, Y=** 1080
17.2.6. ***Screen Match Mode*** : Match Width or Height
17.2.7. ***Match Height*** : 1



Vous êtes prêts à ajouter le bouton sur le panneau : **CanvasMenu.**

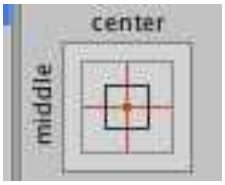
18. Dans la barre de menus, allez dans :

GameObject | UI | Button

19. Sélectionnez le bouton.

19.1. Nommez-le: **Btn_debuter.**

19.2. Dans *l'Inspector,* changez les paramètres suivants :



Rect Transform
19.2.1. ***Anchor Presets*** **:** ancrage **Middle | Center**

19.2.2. ***Pos :*** **X** = 0**, Y =** -300**, Z =** 0
(Il est préférable de garder ce paramètre pour la fin).

19.2.3. ***Width*** **:** 305**,** ***Height*** **:** 125
19.2.4. ***Anchors Min :*** **X**=0.5, **Y**=0.5
19.2.5. ***Anchors Max*** **: X**=0.5, **Y**=0.5
19.2.6. ***Pivot*** **: X**=0.5, **Y**=0.5

Image (Script)
19.2.7. ***Source Image*** **:** btn_bg
19.2.8. ***Color*** **:** blanc
19.2.9. ***Material*** **:** Aucun
19.2.10. ***Image Type*** **:** Sliced
19.2.11. ***Fill Center*** **:** oui

Button (Script)
19.2.12. ***Interactable*** : oui
19.2.13. ***Transition*** **:** Color Tint
19.2.14. ***Normal Color*** **:** couleur de votre choix, par défaut
19.2.15. ***Highlighted Color*** **:** couleur de votre choix, pour la surbrillance.
19.2.16. ***Pressed Color*** **:** couleur de votre choix, quand le bouton est appuyé.
19.2.17. ***Disabled Color*** **:** gris
19.2.18. ***Color Multiplier*** **:** 1
19.2.19. ***Fade Duration*** **:** 0.1

Vous allez ajouter une ombre portée, en-dessous du bouton, afin de le rendre plus visible.

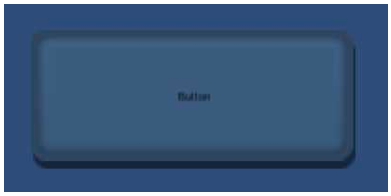
20. Appuyez sur le bouton « Add Component », puis :

UI | Effects | Shadow

20.1. Changez les paramètres de l'effet pour :

Shadow (Script)
20.1.1. ***Effect Color*** **:** Noir
20.1.2. ***Effect Distance*** **: X**= 2**, Y** = -8
20.1.3. ***Use Graphic Alpha*** **:** oui

Votre bouton devrait ressembler à ceci : (la couleur peut varier).



Sauvegardez votre scène. (CTRL+S) ou **(⌘+S)** sur Mac.

Vous allez changer le texte du bouton.

21. Sélectionnez l'élément : **Text** qui se trouve dans le bouton : **Btn_debuter**.

21.1. Dans *l'Inspector,* changez les paramètres :

Text (Script)
21.1.1. ***Text*** : Débuter

Character
21.1.2. ***Font*** : CharisSILMali-B
21.1.3. ***Font Style*** : Normal
21.1.4. ***Font Size*** : 42
21.1.5. ***Line Spacing*** : 1
21.1.6. ***Rich Text*** : oui

Paragraph
21.1.7. ***Alignment*** **:** Centré, Milieu
21.1.8. ***Horizontal Overflow*** : Overflow
21.1.9. ***Vertical Overflow*** : Overflow
21.1.10. ***Best Fit*** : Non
21.1.11. ***Color*** : Blanc
21.1.12. ***Material*** : Aucun

21.2. Appuyez sur le bouton « Add Component », puis :

UI | Effects | Outline

21.3. Changez les paramètres suivants :

Outline (Script)
21.3.1. ***Effect Color*** **:** Noir
21.3.2. ***Effect Distance*** **: X**= 3**, Y** = -3
21.3.3. ***Use Graphic Alpha*** **:** oui

Sauvegardez votre scène. (CTRL+S) ou **(⌘+S)** sur Mac.

Testez votre jeu.

Vous devriez avoir un bouton qui change de couleur en passant au-dessus et qui reste toujours dans le bas de l'écran, peu importe la résolution ou la taille de celui-ci.

Retournez en mode édition.

Maintenant que vous avez un bouton pour démarrer le jeu, vous allez mettre le logo du jeu au centre de l'écran.

Ajout d'un logo dans le menu

22. Sélectionnez : **CanvasMenu** dans *Hierarchy*.

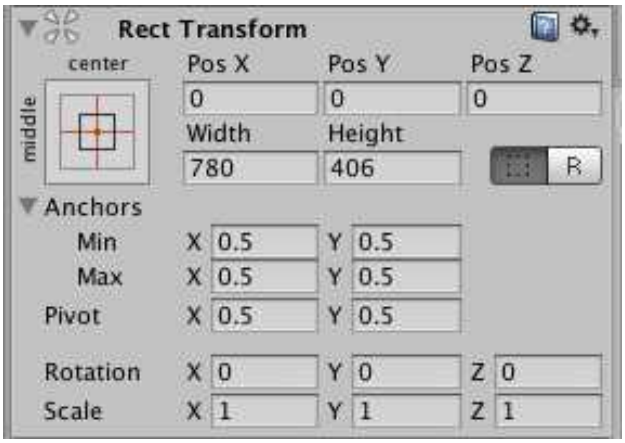
23. Dans la barre de menus, allez dans :

GameObject | UI | Image

24. Sélectionnez l'image.

24.1. Nommez l'image: **Bgr_match2.**

24.2. Changez les paramètres de *l'Inspector* :



Rect Transform
24.2.1. ***Pos :*** **X** = 0**, Y =** 0**, Z =** 0
(Il est préférable de garder ce paramètre pour la fin).

24.2.2. ***Anchors :*** **Min : X**=0.5, **Min Y**=0.5
24.2.3. ***Anchors :*** **Max : X**=0.5, **Max Y**=0.5
24.2.4. ***Pivot*** **: X**=0.5, **Y**=0.5

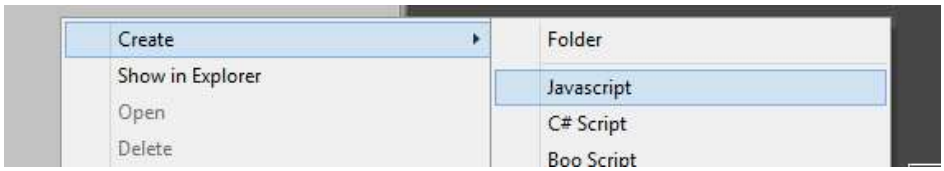
Image (Script)
24.2.5. ***Source Image*** **:** SplashScreen
24.2.6. ***Color*** **:** blanc
24.2.7. ***Material*** **:** Aucun
24.2.8. ***Image Type*** **:** Simple
24.2.9. ***Preserve Aspect*** : oui

25. Appuyez sur le bouton « Set Native Size » pour ajuster automatiquement la taille de l'image.

Vous devriez avoir une interface complète pour votre petit jeu. Il faut maintenant ajouter un script afin de charger votre scène **game** quand on appuie sur le bouton « débuter ».

26. Créez un script en faisant un clic secondaire dans le panneau *Project,* puis :

- **Create > Javascript**



26.1. Nommez ce script : **btn_debuter.**

26.2. Glissez ce script sur l'objet **: EventSystem** dans *Hierarchy.*

Ouvrez le script « btn_debuter.js ».

Ajoutez les lignes suivantes :

```
function OnClick(){
 Application.LoadLevel("game");
}
```

Sauvegardez le script et retournez dans l'éditeur.

En détails…

L'instruction (**Application.LoadLevel**) charge la scène qui est spécifiée entre guillemets. Assurez-vous que le nom de la scène correspond bien à celui de celle-ci « **game** ». N'oubliez-pas que « **Game** », « **game** » ou « **GAME** » sont considérés comme de différents paramètres.

Son nom doit être exact.

27. Sélectionnez le bouton : **Btn_debuter.**

27.1. Dans la section ***On Click ()*** de votre *Inspector,* appuyez sur (**+**) puis changez :

Editor And Runtime

27.1.1. Glissez l'objet **EventSystem** dans le champ du bas.

27.1.2. Dans le champ de droite, sélectionnez la fonction :

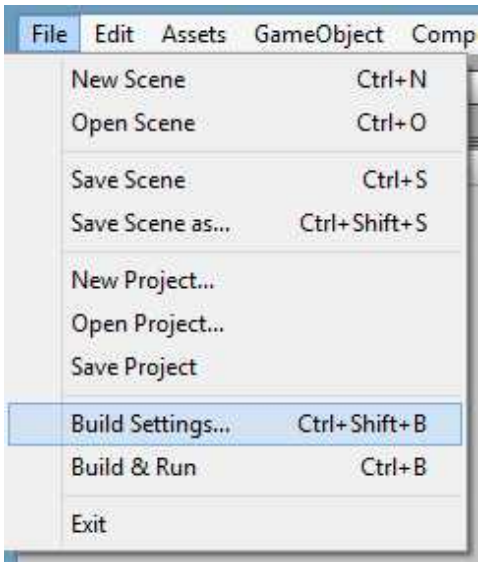
btn_debuter | OnClick()

Pour que le script fonctionne, les scènes doivent se retrouver dans la liste des paramètres du ***build***.

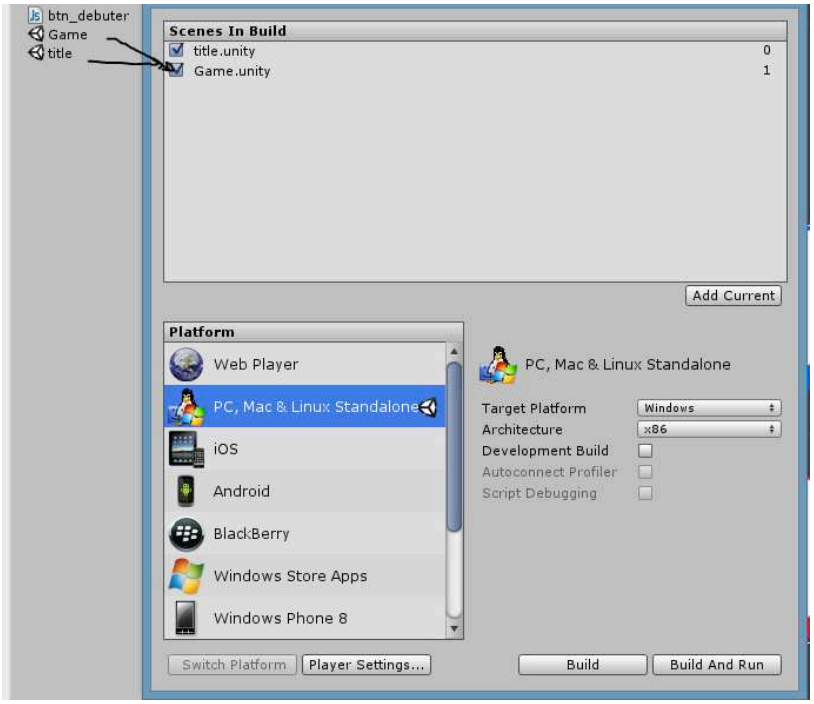
28. Dans la barre de menus, ouvrez :

File | Build Settings…

Raccourcis : (**CTRL+⇧+B**) ou (**⌘+⇧+B**) sur Mac

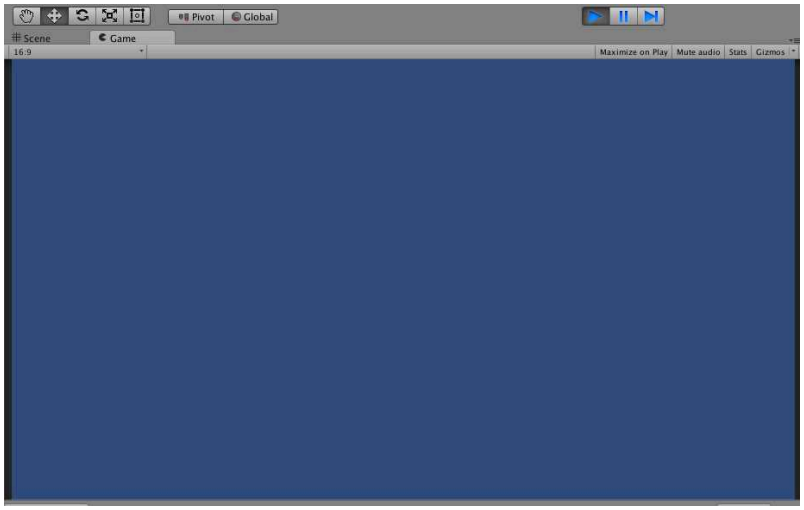


29. Glissez les scènes : **title** et **game** dans la liste et dans cet ordre.



Sauvegardez votre scène. (CTRL+S) ou **(⌘+S)** sur Mac.

Testez votre jeu.



Si vous appuyez sur le bouton « débuter », la scène va changer pour la scène (vide) : **game**.

↘ *La prochaine scène de jeu*

Retournez en mode édition.

Voici un dernier détail, vous allez ajuster la couleur de l'arrière-plan de la caméra.

30. Sélectionnez la caméra : **Main Camera.**

30.1. Dans *l'Inspector*, changez :

Camera
30.1.1. ***Clear Flags*** : Solid Color
30.1.2. ***Background*** : La couleur de votre choix

Sauvegardez votre scène (CTRL+S) ou **(⌘+S)** sur Mac.

Préparation de la scène de jeu

Dans cette partie de l'exercice, vous allez vous consacrer au jeu Match 2. Vous allez créer un panneau centré dans un canevas qui vous servira de planche de jeu. Par la suite, un script va copier et positionner seize cartes en quatre colonnes et quatre rangées.

Débutez par le canevas de la scène.

1. Ouvrez la scène : **game.**

2. Dans la barre de menus, allez dans :

GameObject | UI | Canvas

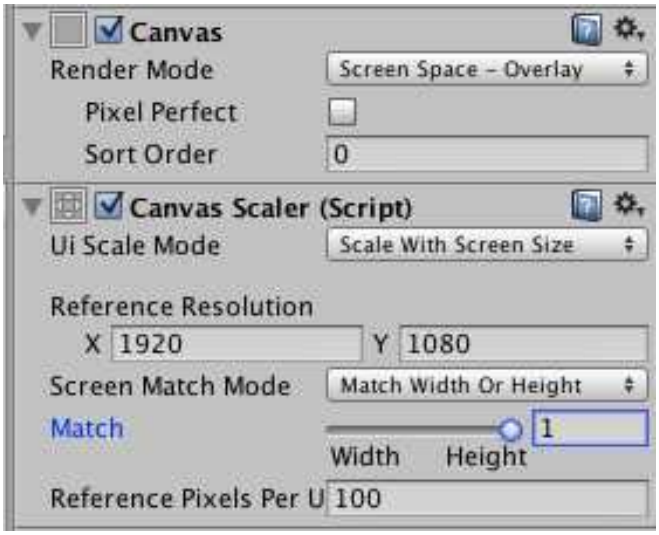
3. Sélectionnez **Canvas.**

3.1. Nommez-le : **CanvasGame.**

3.2. Changez les paramètres suivants :

Canvas
3.2.1. ***Render Mode*** : Screen Space – Overlay
3.2.2. ***Pixel Perfect*** : non
3.2.3. ***Sort Order*** : 0

Canvas Scaler (Script)
3.2.4. ***UI Scale Mode*** : Scale With Screen Size
3.2.5. ***Reference Resolution*** **: X**= 1920**, Y**= 1080
3.2.6. ***Screen Match Mode*** : Match Width or Height
3.2.7. ***Match Height*** = 1



Ajoutez un panneau au centre de ce canevas.

4. Dans la barre de menus, allez dans :

GameObject | UI | Panel

5. Sélectionnez le nouveau **Panel.**

5.1. Dans *l'Inspector,* changez les paramètres suivants :

Rect Transform
5.1.1. ***Pos* : X=** -261**, Y=** -261, **Z=** 0
(Il est préférable de garder ce paramètre pour la fin).

5.1.2. ***Width* =** 522**, *Height* =** 522
5.1.3. ***Anchors :* Min X=** 0.5, **Min Y=** 0.5
5.1.4. ***Anchors :* Max X=**0.5, **Max Y=**0.5
5.1.5. ***Pivot* : X=** 0, **Y=** 0

Vous êtes prêts à ajouter une première carte sur : **Panel.**

6. Sélectionnez le panneau : **CanvasGame.**

6.1. Allez dans la barre de menus :

GameObject | UI | Button

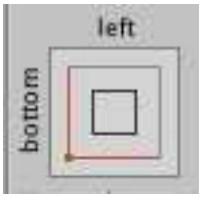
7. Sélectionnez **Button.**

7.1. Nommez le bouton : **Btn_card.**

7.2. Changez la parenté de : **Btn_card**, celui-ci doit être enfant de : **Panel**.

7.3. Dans *l'Inspector*, changez les paramètres suivants :

Rect Transform
7.3.1. ***Anchor Presets* :** ancrage **Bottom | Left**



7.3.2. ***Pos* : X**= 2, **Y=** 2**, Z=** 0
(Il est préférable de garder ce paramètre pour la fin).

7.3.3. ***Width* :** 128**, *Height* :** 128
7.3.4. ***Anchors :* Min X**=0, **Min Y**=0
7.3.5. ***Anchors :* Max X**=0, **Max Y**=0
7.3.6. ***Pivot* : X**=0, **Y**=0

Image (Script)
7.3.7. ***Source Image* :** card_back
7.3.8. ***Color* :** blanc
7.3.9. ***Material* :** Aucun
7.3.10. ***Image Type* :** Simple
7.3.11. ***Preserve Aspect* :** oui

Button (Script)
7.3.12. ***Interactable*** : oui
7.3.13. ***Transition*** **:** Color Tint
7.3.14. ***Normal Color*** **:** couleur de votre choix, par défaut
7.3.15. ***Highlighted Color*** **:** couleur de votre choix, pour la surbrillance
7.3.16. ***Pressed Color*** **:** couleur de votre choix, quand le bouton est appuyé.
7.3.17. ***Disabled Color*** **:** gris
7.3.18. ***Color Multiplier*** **:** 1
7.3.19. ***Fade Duration*** **:** 0.1

Vous allez ajouter une ombre portée en-dessous du bouton pour le rendre plus visible.

8. Dans *l'Inspector*, appuyez sur le bouton « Add Component », puis :

UI | Effects | Shadow

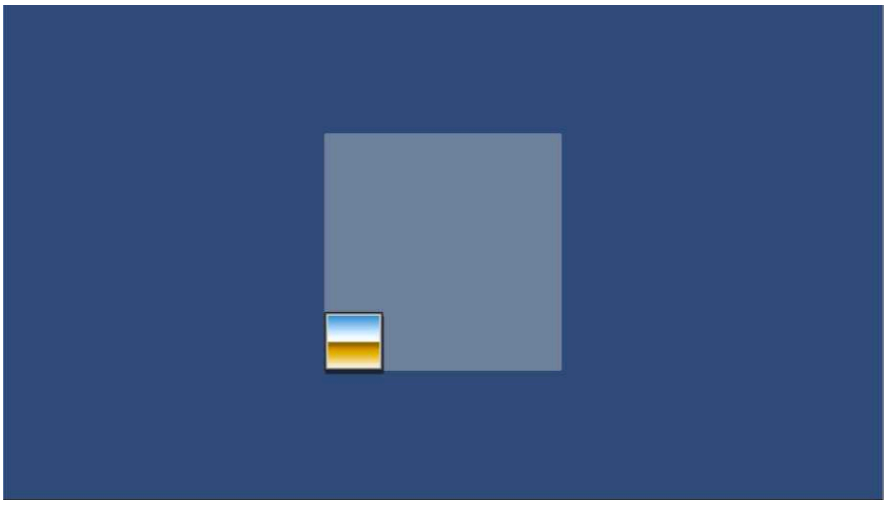
8.1. Changez les paramètres suivants :

Shadow (Script)
8.1.1. ***Effect Color*** **:** Noir
8.1.2. ***Effect Distance*** **: X**= 2**, Y** = -8
8.1.3. ***Use Graphic Alpha*** **:** oui

Vous allez maintenant retirer le texte du bouton.

9. Sélectionnez l'élément **Text** qui se trouve dans le bouton : **Btn_card.**

Supprimez-le en appuyant au clavier sur (**Supprim.**) ou (**⌘+⌫**) sur Mac.



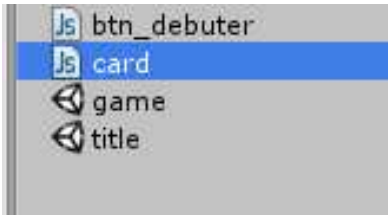
Le résultat devrait ressembler à ceci :

Sauvegardez votre scène (CTRL+S) ou **(⌘+S)** sur Mac.

Vous allez maintenant créer deux scripts : le premier script permettra aux cartes de contenir les éléments de base.

10. Créez le script, en faisant un clic secondaire dans le panneau *Project,* puis :

➢ **Create > Javascript**



10.1. Nommez ce script : **card.**

Ouvrez le script « card.js ».

Ajoutez les variables suivantes :

```
private var isFaceUp:boolean = false; //Carte active
private var imgBack:Sprite; //Dos de la carte
private var imgVal:int; //Valeur de la carte
var img:UI.Image; //Lien vers le visuel du bouton
var GameScript:GameObject; //Objet qui contient le script du jeu
var cardsList:Sprite[]; //Toutes les images possibles
```

↘ *Les commentaires (//) sont optionnels, mais ils aident à comprendre le rôle des variables.*

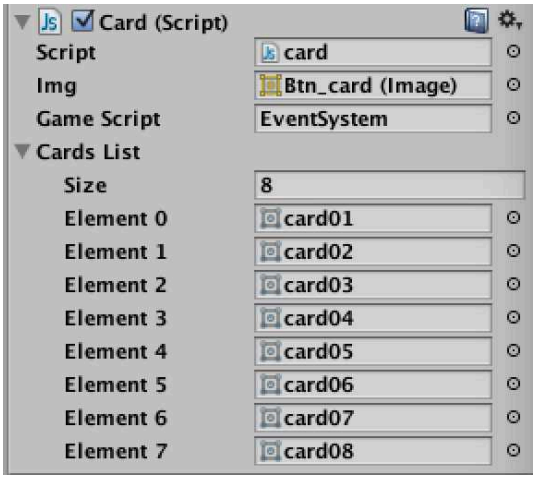
Sauvegardez le script et retournez dans l'éditeur.

11. Glissez le script **card.js** sur le bouton : **Btn_card** dans le panneau *Hierarchy*.

12. Dans votre *Hierarchy*, sélectionnez : **Btn_card.**

12.1. Changez les paramètres suivants dans *l'Inspector* :

Card (Script)
12.1.1. ***Img*** : Glissez **Btn_card** de *Hierarchy* dans cette variable.
12.1.2. ***Game Script*** : EventSystem
12.1.3. ***Cards List*** : **Size** = 8
• ***Element 0*** : card01
• ***Element 1*** : card02
• ***Element 2*** : card03
• ***Element 3*** : card04
• ***Element 4*** : card05
• ***Element 5*** : card06
• ***Element 6*** : card07
• ***Element 7*** : card08



Le script est maintenant prêt à changer le visuel du bouton.

Pour tester les changements, vous allez ajouter une ligne dans le script.

Ouvrez le script « card.js »

Ajoutez les lignes de codes suivantes :

```
…
function Start () {
 imgBack = img.sprite;
 img.sprite = cardsList[0];
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

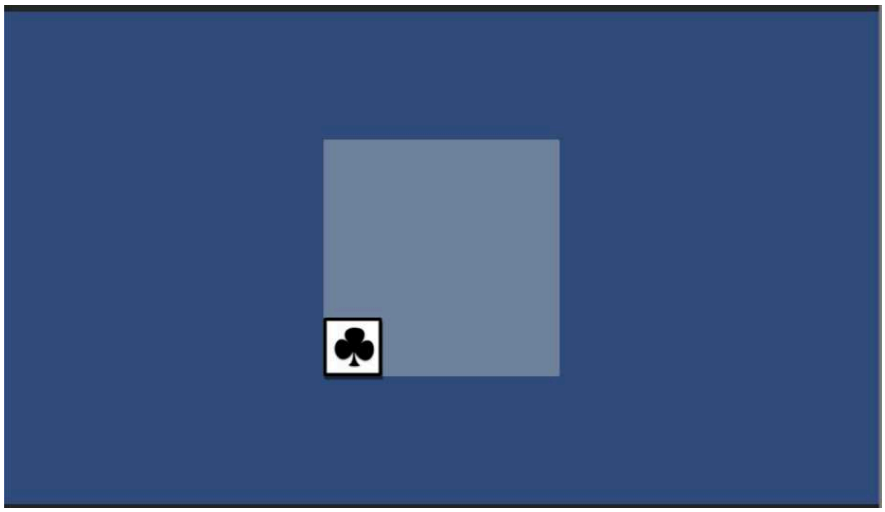
Sauvegardez le script et retournez dans l'éditeur.

En détails…

La première ligne conserve l'image d'origine (le dos de la carte) dans la variable (**imgBack**). La deuxième ligne remplace le visuel de la carte par l'image à **l'élément 0** (**cardsList[0]**).

Sauvegardez votre scène (CTRL+S) ou **(⌘+S)** sur Mac.

Testez votre scène.



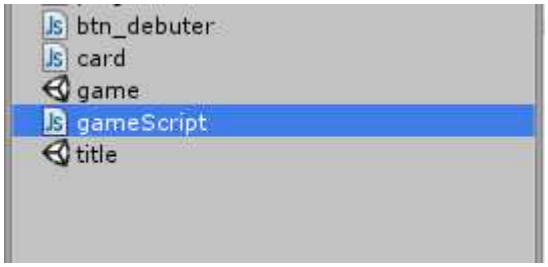
↘ *Une carte avec le visuel :* **card01**

Retournez en mode édition.

Vous allez créer un deuxième script qui va copier le bouton et faire une grille de 4X4 pour le gameplay.

13. Créez un script, en faisant un clic secondaire dans le panneau *Project,* puis :

   - **Create > Javascript**



13.1. Nommez ce script : **gameScript.**

14. Glissez ce script sur l'objet : **EventSystem**.

Les liens permettent présentement aux cartes de communiquer avec le script : **gameScript,** mais c'est impossible de faire le contraire.

Ouvrez le script « gameScript.js ».

Ajoutez les variables suivantes :

```
private var cols :int = 4;
private var rows :int = 4;
private var totalCard :int = cols*rows;
private var matchesNeededToWin :int = totalCard/2;
private var matchesMade :int = 0;
private var cardW :int = 130;
private var cardH :int = 130;
var CardObj :UI.Button;
var panelParent :UI.Image;
```

En détails…

Ces variables vont permettre de dessiner la grille dans le jeu. Nous aurons besoin du nombre de colonnes et de rangées (**cols** and **rows**), du nombre total de cartes (**totalCard**), du nombre de combinaisons nécessaires pour gagner (**matchesNeededToWin**), du nombre de combinaisons réussies (**matchMade**), de la largeur et la hauteur des cartes avec un petit espace entre celles-ci (**cardW** et **cardH**), d'une carte à copier (**CardObj**) et finalement du panneau parent aux cartes (**panelParent**).

Vous allez devoir ajouter une fonction pour dessiner la grille à l'écran.

Toujours dans le même script, ajoutez :

```
…
function Start () {
 drawGrid();
}

function drawGrid(){
 var copyItem:UI.Button;
 for (var k=0;k<cols;k++){
   for (var j=0;j<rows;j++){
     copyItem = Instantiate (CardObj);
     copyItem.transform.SetParent(panelParent.transform,false);
     copyItem.transform.localPosition = Vector3(k*cardW,j*cardH,0);
   }
 }
 CardObj.enabled = false;
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails…

La fonction (**drawGrid()**) crée un clone (**copyItem**) de la carte de références (**Instantiate(CardObj)**), ensuite elle boucle pour chaque colonne (**for : k<cols**) et chaque rangée (**for : j<rows**). En second lieu, on place le clone enfant du panneau (**SetParent**) et on le positionne selon son emplacement dans la grille (**localPosition**).

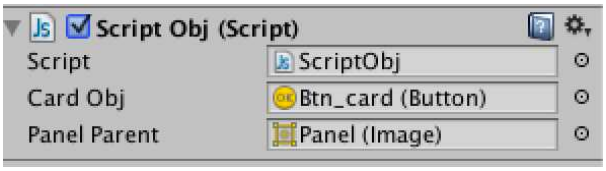
Finalement, on cache la carte de références lorsque la grille est activée (**CardObj.active**).

Vous devez assigner les bons objets dans les variables de ce script.

15. Sélectionnez l'objet : **EvenSystem.**

15.1. Glissez les objets suivants dans les variables :

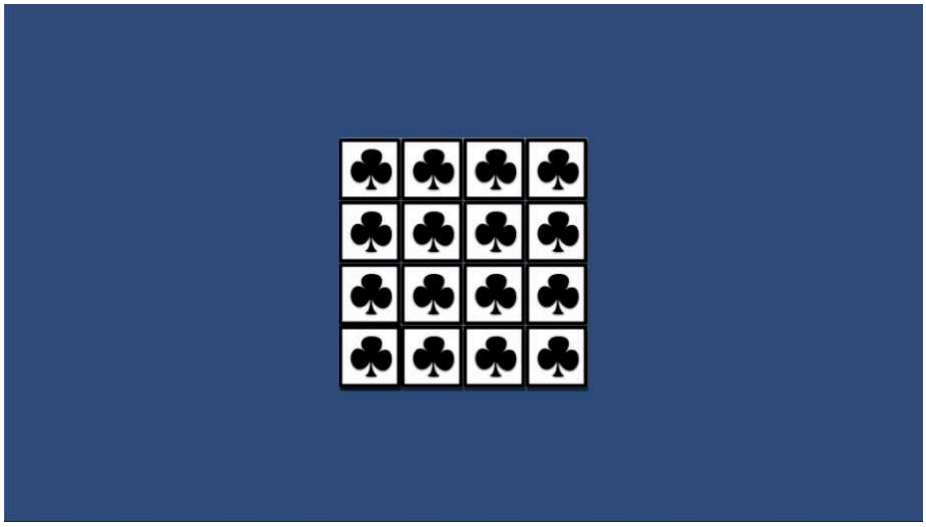
Script Obj (Script)
15.1.1. ***Card Obj*** : glissez **Btn_card** de votre *Hierarchy*
15.1.2. ***Panel Parent*** : glissez **Panel** de votre *Hierarchy*



Sauvegardez votre scène (CTRL+S) ou **(⌘+S)** sur Mac.

Testez votre scène.

Une grille de 4x4 avec un seul modèle de carte devrait s'afficher au centre de l'écran.



Retournez en mode édition.

En détails...

Maintenant que la grille s'affiche correctement et que les boutons fonctionnent, il faut modifier le script : **card.js** pour recevoir le nom de la carte qui se trouve dans le script : **gameScript.js**. Par défaut, nous allons mettre le visuel, sur toutes les cartes.

Ouvrez le script « card.js »

Ajoutez les lignes suivantes :

```
function Start () {
 imgBack = img.sprite;
 //img.sprite = cardsList[0];
}

function showImg(cardNum:int){
 imgVal = cardNum;
 img.sprite = cardsList[imgVal];
}
```

↘ *Les textes en **gras** sont à ajouter ou ~~**retirer**~~ le reste demeure inchangé.*

Sauvegardez le script et retournez dans l'éditeur.

En détails…

Les changements font en sorte que l'on n'affiche plus la carte de trèfle par défaut. La fonction (**showImg**) assigne les valeurs de l'image qui doit être affichée.

Nous allons ajouter au script : **gameScript.js,** les instructions afin de créer une liste de toutes les cartes. Nous allons également assigner une valeur de carte de façon aléatoire durant la création de celle-ci.

Ouvrez le script « gameScript .js »

Ajoutez les lignes suivantes :

```
…
var panelParent :UI.Image;
private var aCards:Array = new Array();
private var aGrid:Array = new Array();
private var aCardsFlipped:Array = new Array();
private var canClick:boolean = true;

…
function drawGrid(){
   canClick = true;
   var randomElem:int;
   var imgVal:int;
   for (var i:int = 0; i<matchesNeededToWin;i++){
     aGrid.Add(i);
     aGrid.Add(i);
   }
   var copyItem:UI.Button;
   for (var k=0;k<cols;k++){
    for (var j=0;j<rows;j++){
      randomElem = Random.Range(0, aGrid.length);
      imgVal = aGrid[randomElem];
      aGrid.RemoveAt(randomElem);
      copyItem = Instantiate (CardObj);
      copyItem.transform.SetParent(panelParent.transform,false);
      copyItem.transform.localPosition = Vector3(k*cardW,j*cardH,0);
      copyItem.SendMessage("showImg", imgVal);
    }
   }
   CardObj.active = false;
}
```

*Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

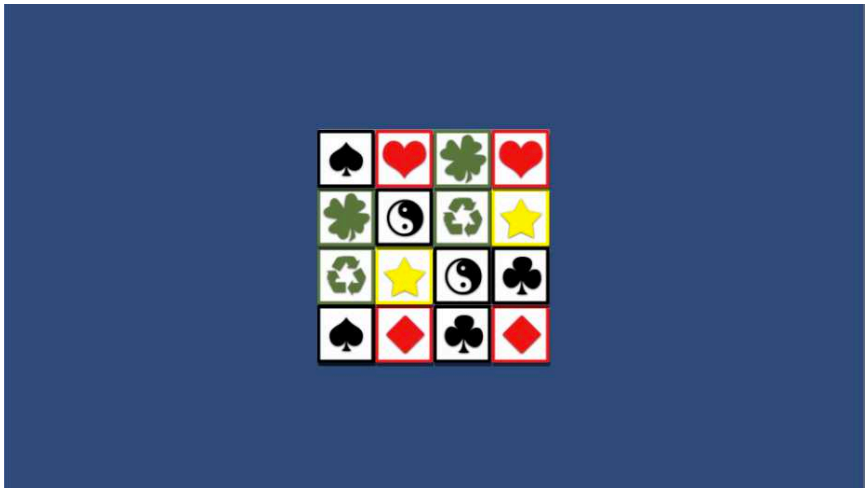
Sauvegardez le script et retournez dans l'éditeur.

En détails...

Nous avons maintenant trois listes : la première contient une liste de toutes les cartes possibles (**aGrid**), la deuxième, une liste des cartes sélectionnées (**aCards**) finalement, une liste des cartes retournées. La variable (**canClick**) permet d'empêcher le joueur de faire une sélection après que deux cartes aient été choisies. La fonction (**drawGrid**) crée une liste ordonnée de toutes les cartes en double. Puis, lors du clonage des cartes, on assigne de façon aléatoire une valeur à cette nouvelle carte (**SendMessage("showImg", imgVal)**) et on retire la carte pigée de la liste (**aGrid.RemoveAt**).

Testez le jeu.

Toutes les combinaisons sont maintenant assignées aux différentes cartes.



Retournez en mode édition.

Vous allez ajouter deux fonctions et changer quelques commentaires pour retourner les cartes.

Ouvrez le script « card.js »

Ajoutez les lignes suivantes :

```
function showImg(cardNum:int){
 imgVal = cardNum;
 //img.sprite = cardsList[imgVal];
}

function flip(){
 if (isFaceUp){
  img.sprite = imgBack;
 }else{
  img.sprite = cardsList[imgVal];
 }
 isFaceUp = !isFaceUp;
}
```

↘ *Les textes en **gras** sont à ajouter ou ~~retirer~~ le reste demeure inchangé.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Désormais, on n'affiche plus la valeur de la carte dès le début du jeu, mais seulement quand on appelle la fonction **flip().** Celle-ci change l'image de la carte entre le revers et la carte affichée.

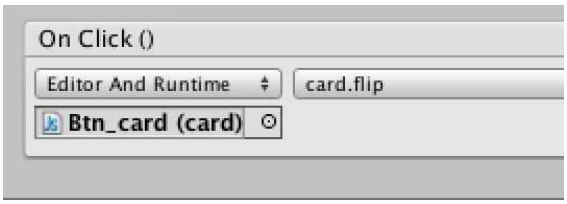
16. Sélectionnez : **Btn_card.**

16.1. Dans la section : ***On Click ()*** appuyez sur le bouton (**+**).

16.2. Dans votre *Inspector,* changez :

16.2.1. Glissez **Btn_card** dans la case du bas.
16.2.2. Dans la case de droite, sélectionnez la fonction :

card | flip()

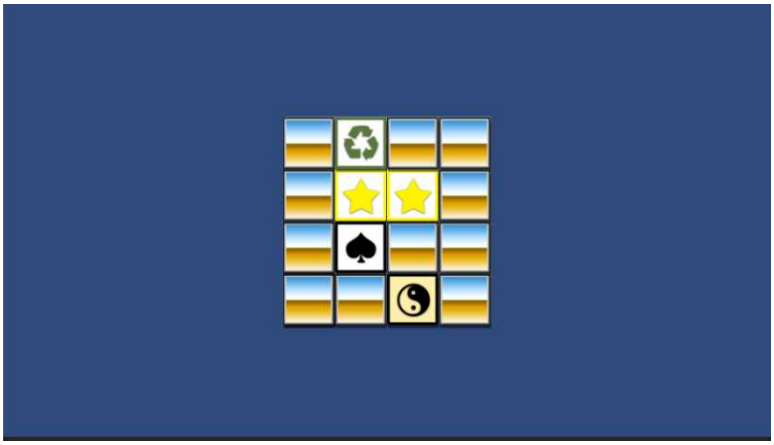


Lorsque l'on appuie sur le bouton, celui-ci connaît quelle fonction appeler.

Sauvegardez votre scène (CTRL+S) ou **(⌘+S)** sur Mac.

Testez le jeu.

Vous pouvez maintenant jouer au jeu, mais aucun règlement ne permet de limiter les actions du joueur ou de valider les résultats.



Retournez en mode édition.

La prochaine étape est de limiter le nombre de cartes que l'on peut retourner à la fois.

Ouvrez le script « gameScript.js ».

Ajoutez les lignes suivantes :

```
…
function clicOnCard(obj:GameObject){
 if (canClick){
  aCardsFlipped.Add(obj);
  obj.SendMessage("turn");
  if (aCardsFlipped.length == 2){
   canClick = false;
  }
 }
}
…
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails…

Si deux cartes sont activées (**aCardsFlipped.length**), nous ne pouvons plus en retourner d'autres (**canClick**).

Dorénavant, ce sera cette fonction qui se chargera de retourner les cartes.

Ouvrez le script « card.js ».

Ajoutez les lignes suivantes :

```
…
function flip(){
 if (isFaceUp){
   img.sprite = imgBack;
 }else{
   img.sprite = cardsList[imgVal];
 }
 isFaceUp = !isFaceUp;
 if (!isFaceUp){
   GameScript.SendMessage("clicOnCard", gameObject);
 }
}

function turn(){
  if (isFaceUp){
    img.sprite = imgBack;
  }else{
    img.sprite = cardsList[imgVal];
  }
    isFaceUp = !isFaceUp;
}
```

↘ *Les textes en **gras** sont à ajouter ou ~~**retirer**~~ le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

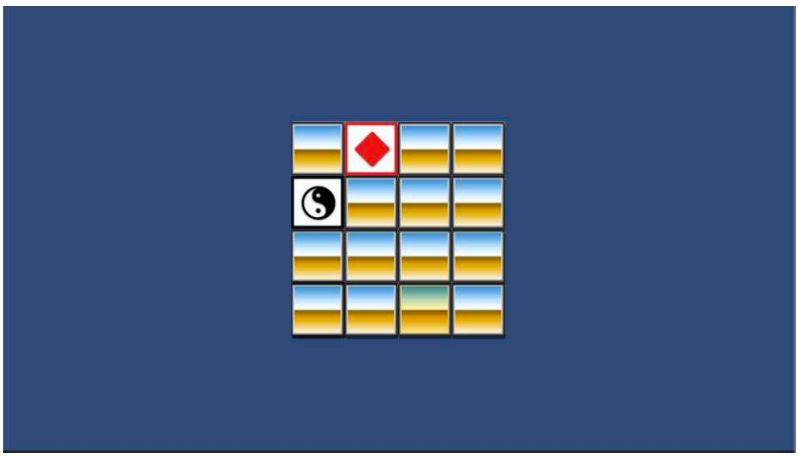
En détails…

Nous venons de modifier le script des cartes. La fonction (**flip()**) validera si la carte est cachée. Si oui, la fonction (**clicOnCard()**) est appelée pour s'assurer que moins de deux cartes sont actives. Si la condition est vraie, alors on active cette carte avec la fonction (**turn()**).

Testez le jeu.

Désormais, seulement 2 cartes peuvent être retournées à la fois.

Retournez en mode édition.



Ouvrez le script « gameScript.js ».

Ajoutez les lignes suivantes :

```
…
function clicOnCard(obj:GameObject){
  if (canClick){
    aCardsFlipped.Add(obj);
    obj.SendMessage("turn");
    if (aCardsFlipped.length == 2){
      canClick = false;
      validate();
    }
  }
}
…
function setValCard(num:int){
   aCards.Add(num);
}
function validate(){
  if (aCards[0] == aCards[1]){
     matchesMade++;
     aCardsFlipped.Clear();
     aCards.Clear();
     canClick = true;
  }else{
     Invoke ("flipCards", 1);
  }
}
function flipCards(){
  var obj:GameObject = aCardsFlipped[0];
  obj.SendMessage("turn");
  obj = aCardsFlipped[1];
  obj.SendMessage("turn");
  aCardsFlipped.Clear();
  aCards.Clear();
  canClick = true;
}
…
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script.

En détails…

Nous venons d'ajouter 3 fonctions à ce script : une première pour mémoriser le numéro des cartes choisies (**setValCard**), une seconde pour valider les cartes (**validate**) et finalement une dernière pour réinitialiser les cartes quand elles ne sont pas identiques (**flipCards**).

Voici la manipulation qui servira à mémoriser la valeur de la carte :

Ouvrez le script « card.js ».

Ajoutez les lignes suivantes :

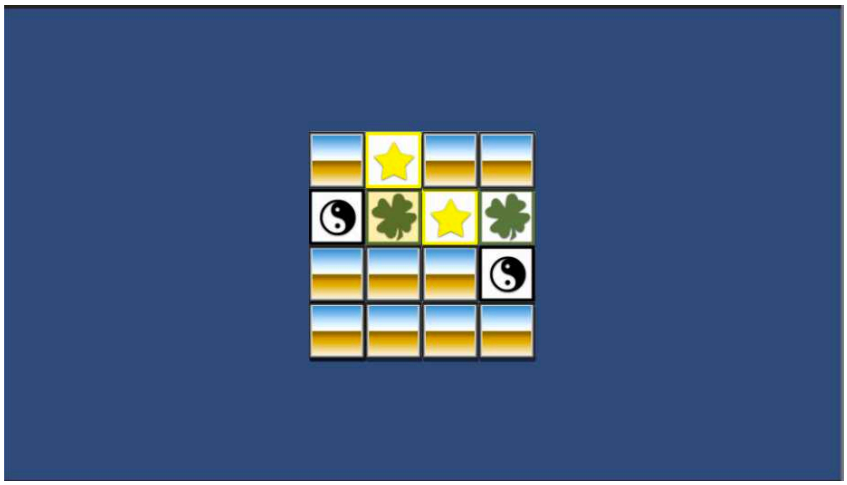
```
…
function flip(){
 if (!isFaceUp){
  GameScript.SendMessage("setValCard", imgVal);
  GameScript.SendMessage("clicOnCard", gameObject);
 }
}
…
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

Testez le jeu.

Vous pouvez maintenant jouer au jeu, au complet.



Retournez en mode édition.

Vous allez ajouter du code pour relancer le jeu automatiquement quand vous complètez le puzzle.

Ouvrez le script « gameScript.js ».

Ajoutez les lignes suivantes :

```
…

function validate(){
  if (aCards[0] == aCards[1]){
     matchesMade++;
     if (matchesMade < matchesNeededToWin){
       aCardsFlipped.Clear();
       aCards.Clear();
       canClick = true;
     }else{
       Invoke ("victory", 1);
     }
  }else{
     Invoke ("flipCards", 1);
  }
}

function victory(){
  Application.LoadLevel("game");
}
…
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

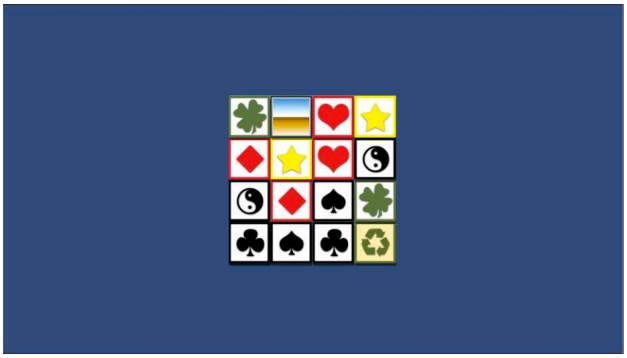
Sauvegardez le script et retournez dans l'éditeur.

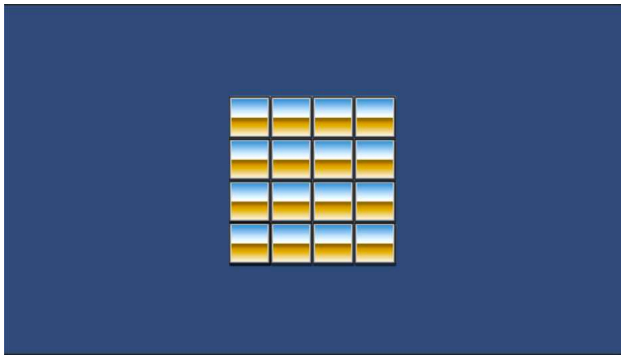
En détails…

Nous venons de mettre une limite au gameplay. Si le nombre de pairs est plus petit que le nombre maximal, alors on continue. Sinon, on appelle la fonction **victory()** après une seconde (**Invoke**). La fonction **victory()** est celle qui recharge le niveau.

Sauvegardez votre scène (CTRL+S) ou **(⌘+S)** sur Mac.

Testez le jeu.





Retournez en mode édition.

Voilà, c'est complet ! Retournez à la scène **title** et vous aurez l'expérience totale.

> **Conseil!**
>
> Maintenant, tentez d'exporter votre jeu en version redistribuable comme vous l'avez fait dans le chapitre 4.

Codes complétés

Voici les codes complets non abrégés de l'exercice :

card

```
#pragma strict

private var isFaceUp:boolean = false; //Carte active
private var imgBack:Sprite; //Dos de la carte
private var imgVal:int; //Valeur de la carte
var img:UI.Image; //Lien vers le visuel du bouton
var GameScript:GameObject; //Objet qui contient le script du jeu
var cardsList:Sprite[]; //Toutes les images possibles

function Start(){
  imgBack = img.sprite;
}

function showImg(cardNum:int){
  imgVal = cardNum;
}

function flip(){
  if (!isFaceUp){
     GameScript.SendMessage("setValCard", imgVal);
     GameScript.SendMessage("clicOnCard", gameObject);
  }
}

function turn(){
  if (isFaceUp){
     img.sprite = imgBack;
  }else{
     img.sprite = cardsList[imgVal];
  }
  isFaceUp = !isFaceUp;
}
```

gameScript

```
#pragma strict

private var cols :int=4;
private var rows :int=4;
private var totalCard :int=cols*rows;
private var matchesNeededToWin :int=totalCard/2;
private var matchesMade :int=0;
private var cardW :int=130;
private var cardH :int=130;
var CardObj :UI.Button;
var panelParent :UI.Image;
private var aCards:Array=new Array();
private var aGrid:Array=new Array();
private var aCardsFlipped:Array=new Array();
private var canClick:boolean=true;

function Start () {
    drawGrid();
}

function drawGrid(){
  canClick = true;
  var randomElem:int;
  var imgVal:int;
  for (var i:int=0;i<matchesNeededToWin;i++){
     aGrid.Add(i);
     aGrid.Add(i);
  }
  var copyItem:UI.Button;
  for (var k=0;k<cols;k++){
     for (var j=0;j<rows;j++){
       randomElem = Random.Range(0,aGrid.length);
       imgVal = aGrid[randomElem];
       aGrid.RemoveAt(randomElem);
       copyItem = Instantiate (CardObj);
       copyItem.transform.SetParent(panelParent.transform,false);
       copyItem.transform.localPosition = Vector3(k*cardW,j*cardH,0);
       copyItem.SendMessage("showImg", imgVal);
     }
  }
  CardObj.enabled = false;
}
```

gameScript (fin)

```
function clicOnCard(obj:GameObject){
  if (canClick){
     aCardsFlipped.Add(obj);
     obj.SendMessage("turn");
     if (aCardsFlipped.length == 2){
        canClick = false;
        validate();
     }
  }
}

function setValCard(num:int){
  aCards.Add(num);
}

function validate(){
  if (aCards[0] == aCards[1]){
  matchesMade++;
     if (matchesMade < matchesNeededToWin){
        aCardsFlipped.Clear();
        aCards.Clear();
        canClick = true;
     }else{
        Invoke ("victory", 1);
     }
  }else{
     Invoke ("flipCards", 1);
  }
}

function victory(){
  Application.LoadLevel("game");
}

function flipCards(){
  var obj:GameObject = aCardsFlipped[0];
  obj.SendMessage("turn");
  obj = aCardsFlipped[1];
  obj.SendMessage("turn");
  aCardsFlipped.Clear();
  aCards.Clear();
  canClick = true;
}
```

unity

Chapitre 7 - L'éditeur de terrain

Matière couverte dans ce chapitre

- **Création du relief de la scène**
- **Ajout des textures**
- **Création d'arbres**
- **Ajout de la végétation**
- **Ajout d'une étendue d'eau**
- **Ajout d'un ciel et d'une atmosphère**
- **Ajout d'effets de lumière**
- **Utilisation d'*Image Effects***

Dans la réalisation de vos environnements, pour le jeu avec Unity® vous avez deux possibilités, premièrement, vous pouvez utiliser un logiciel de création 3D spécialisé comme 3DS Max, Maya, XSI, et Blender ou vous servir des outils de création de terrain inclus.

Idéalement, il est préférable de pouvoir utiliser les deux, car la majorité des jeux contiennent un environnement topographique, meublé avec des objets créés avec des logiciels tiers. De plus, les environnements clos ne peuvent pas être créés avec l'éditeur de terrain.

Nous traiterons uniquement de l'éditeur de terrain inclus avec Unity®.

Atelier pratique

Création de terrains

Les outils de création d'environnement sont très faciles à utiliser et vont vous permettre d'obtenir des résultats convaincants en quelques heures, plutôt que quelques jours. Sans compter que plusieurs scripts seront à votre disposition pour accélérer votre prototypage.

Les éléments de la scène

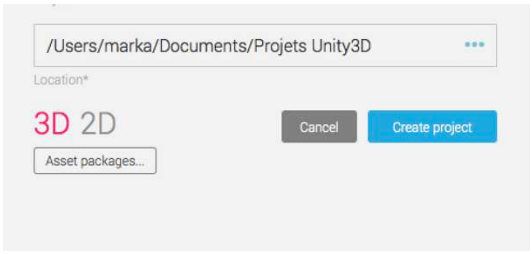
1. Ouvrez Unity®.

2. Cliquez sur le bouton : « New Project ».

3. Sauvez votre projet sur un disque dur local, avec le nom : **Terrain.**

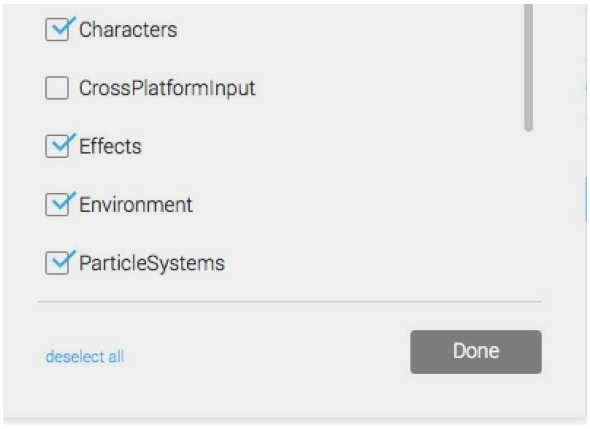
3.1. Ce sera un projet : **3D.**

3.2. Cliquez sur « Asset Packages… »



3.3. Sélectionnez les paquets suivants :

- ***Characters***
- ***Effects***
- ***Environment***
- ***ParticleSystems***



4. Appuyez sur « Done » pour retourner à l'écran de démarrage.

5. Appuyez sur « Create Project » pour débuter.

6. Une fois votre projet ouvert, **sauvegardez votre scène** (**CTRL+S**) ou (**⌘+S**) sur Mac.

6.1. Nommez la scène : **Terrain**.

Vous allez maintenant créer un nouveau terrain vide.

7. Allez dans la barre de menus :

GameObject | 3D Objects | Terrain

Un nouveau terrain devrait apparaître dans *Hierarchy*.

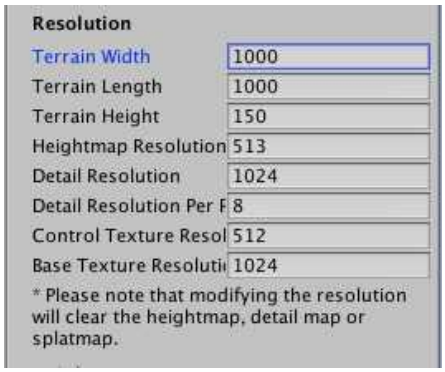
8. Sélectionnez le terrain.

8.1. localisez le ***Component*** : **Terrain (Script)** dans *l'Inspector*.

Cet outil comprend sept sections. Vous allez d'abord configurer les paramètres du terrain.

9. Cliquez sur la dernière icône de cette liste pour ouvrir les paramètres : ***terrain settings***.

9.1. Dans *l'Inspector*, changez :



Resolution
9.1.1. ***Terrain Width*** : 1000
9.1.2. ***Terrain Length*** : 1000
9.1.3. ***Terrain Height*** : 150

↘ *Laissez les autres paramètres inchangés.*

En détails...

Notez que les dimensions du terrain sont en **mètres** et que le paramètre **Height** définit la hauteur maximale du terrain.

Également, **Heightmap Resolution** définit la taille de la texture 2D qui est utilisé pour la topologie du terrain. Cette texture est en puissance de 2 (128x128, 256x256, 512x512, 1024x1024, etc.) plus 1 pixel pour marquer un « vertex point ». Les dimensions valides sont donc : **129, 257, 513, 1025...**

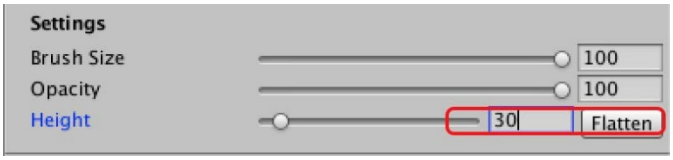
Puisque vous avez dimensionné la taille et la résolution du terrain, vous allez élever le terrain à 30 mètres au-dessus du zéro absolu.

Création du relief

10. Cliquez sur la deuxième icône de la liste pour ouvrir l'outil ***: Paint height.***

Cet outil permet d'élever ou de descendre le terrain à une hauteur fixe.

10.1. Dans les ***settings***, changez :



Settings
10.1.1. ***Height*** : 30

10.2. Appuyez sur le bouton « Flatten ».

En détails...

La scène va s'aplatir à cette hauteur. Faites attention de ne pas appuyer sur ce bouton par accident, durant la création de la scène.

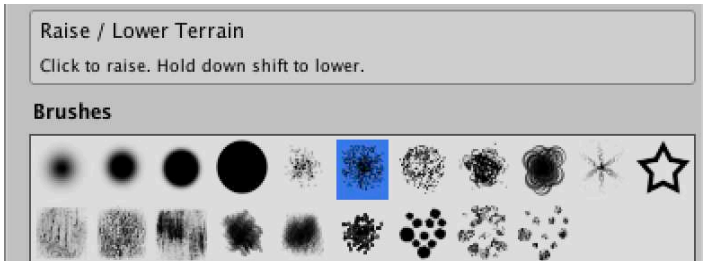
Vous allez changer la topologie afin de créer une île entourée d'eau.

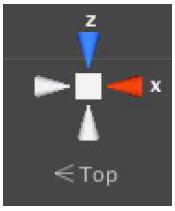
11. Cliquez sur la première icône de l'outil de terrain pour ouvrir ***Raise / Lower Terrain***.

La différence avec cet outil, c'est que celui-ci n'a pas de hauteur fixe et peut seulement monter ou descendre une portion de la topologie.

11.1. Dans la section ***settings***, changez :

11.1.1. ***Brushes*** : Particules
11.1.2. ***Brush Size***: 100
11.1.3. ***Opacity*** *:* 75

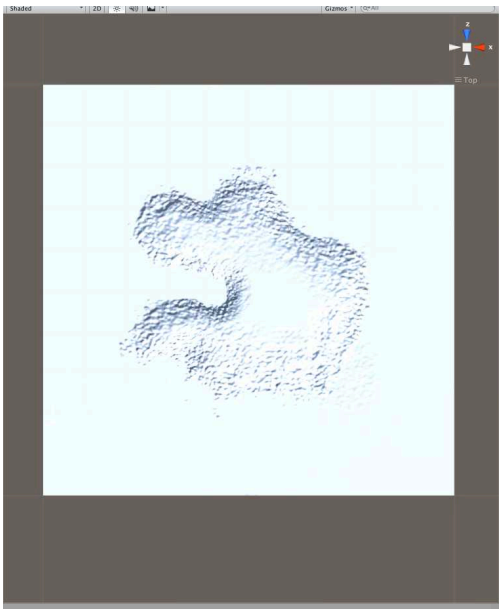




12. Mettez la caméra en vue ***Top,*** en utilisant le ***Gizmo*** dans le coin en haut, à droite de la fenêtre ***Scene.***

13. En-dessous du ***Gizmo,*** il y a une icône à gauche de l'axe. Cliquez dessus afin de changer la perspective de la caméra pour une vue isométrique illustrée par des lignes parallèles.

14. Tenez le bouton (MAJ. ⇧) enfoncé sur votre clavier afin de creuser autour de votre île, comme illustré :



Notez-bien!

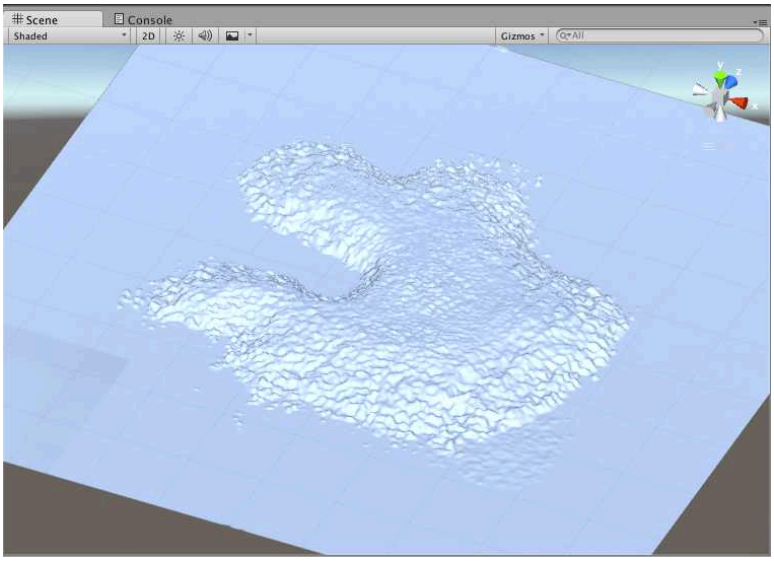
Le point le plus bas auquel votre terrain peut descendre est 0 mètre. Vous devriez voir le sol s'aplatir quand vous aurez descendu à ce niveau.

Conseil!

Pour cette notion, il n'est pas nécessaire de reproduire exactement cette forme mais, elle peut vous servir comme canevas modèle. Également, laissez amplement d'espace autour de l'île afin de ne pas voir la brisure entre le sol et le ciel.

15. En lâchant la touche (⇧) sur votre clavier, ajoutez quelques variations de relief de terrain.

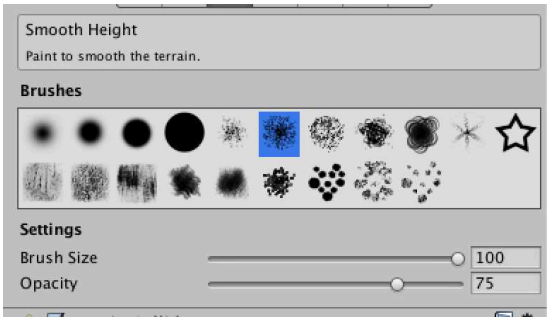
Vous devriez obtenir un terrain chaotique comme ceci, dépendamment du type de brosse que vous utilisez.



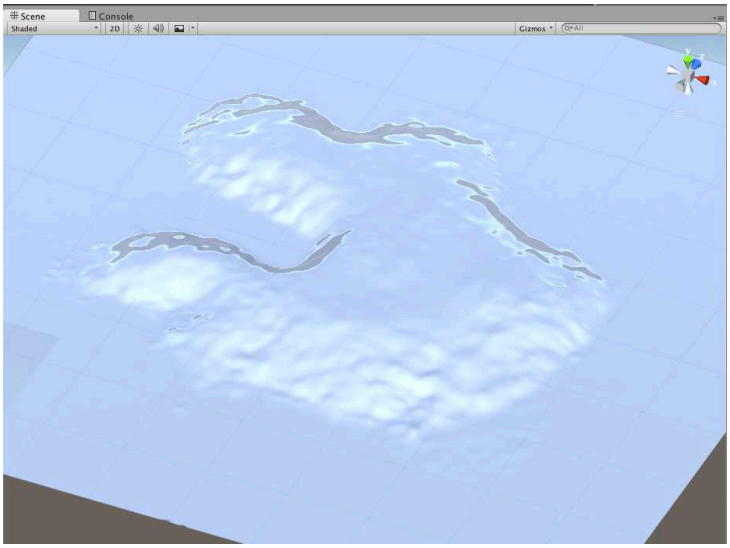
La prochaine étape consiste à adoucir le relief pour le rendre plus organique.

16. Cliquez sur la 3e icône de l'outil terrain pour ouvrir (adoucir le relief) ***Smooth Height***.

16.1. Ajustez (selon vos besoins) les paramètres de l'outil :



16.1.1. ***Brushes*** : Particules
16.1.2. ***Brush Size*** : 100
16.1.3. ***Opacity*** : 75



17. Il est maintenant temps d'adoucir le relief de votre île. Retenez que ce terrain sera vu de près, il est important de ne pas laisser la surface trop chaotique.

Comparez la différence une fois le terrain adouci :

Vous avez la base de l'île, vous allez y ajouter un volcan.

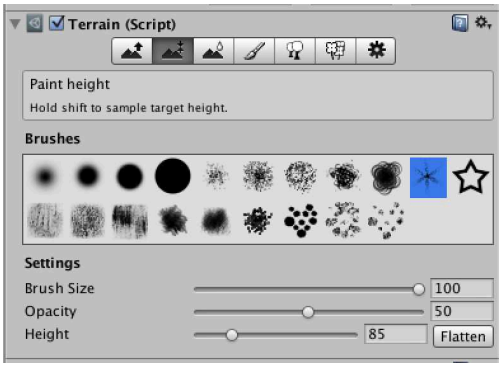
Ajout d'un volcan

Afin de créer ce volcan, vous allez utiliser les outils pour peindre à une hauteur déterminée.

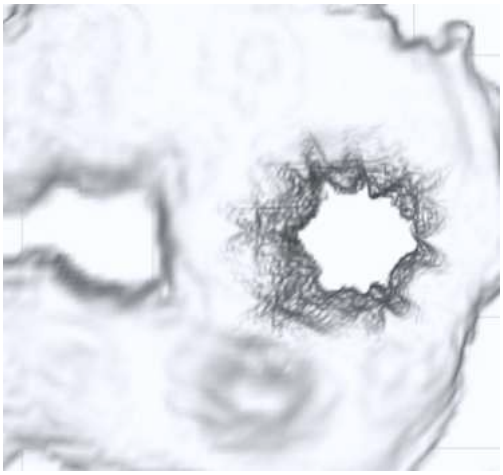
18. Cliquez sur la 2e icône de la liste pour ouvrir : ***Paint height.***

18.1. Changez les paramètres suivants pour colorer le sommet du volcan.

18.1.1. ***Brushes*** : Forme de flocon
18.1.2. ***Brush Size*** : 100
18.1.3. ***Opacity*** : 50
18.1.4. ***Height*** : 85

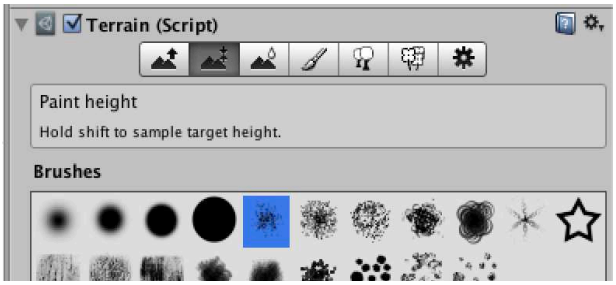


19. Créez un monticule jusqu'à ce que le sommet devienne plat.



Vous avez la forme de base, vous allez creuser l'intérieur du volcan avec une nouvelle hauteur déterminée.

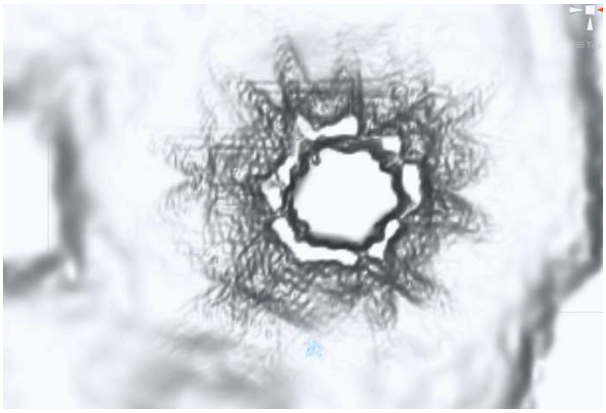
20. Toujours dans la même section :



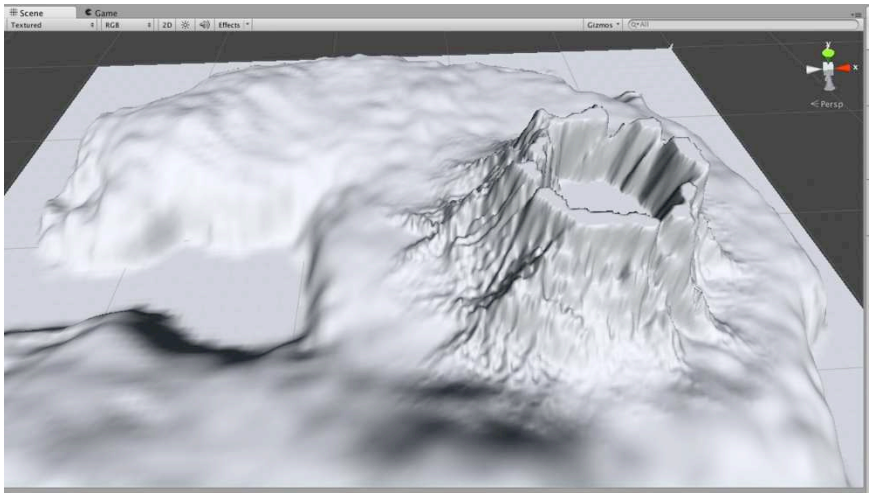
20.1. Changez les paramètres de la brosse pour :

20.1.1. ***Brushes*** : Particules
20.1.2. ***Brush Size*** : 20
20.1.3. ***Opacity*** : 50
20.1.4. ***Height*** : 55

21. Colorez à l'intérieur du volcan comme ceci :

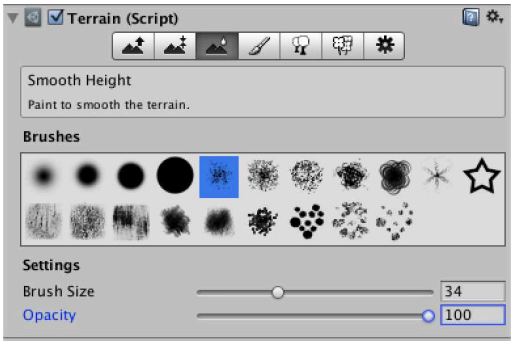


En perspective, vous devriez obtenir ce résultat :



Ensuite, vous allez adoucir les rebords du volcan.

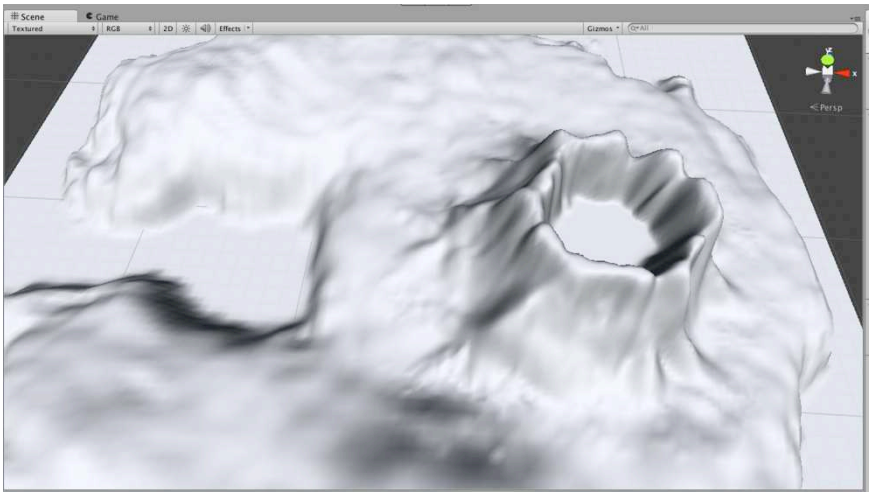
22. Cliquez sur la 3e icône de l'outil terrain : ***Smooth Height***.



22.1. Changez les paramètres suivants :

Brushes : Particules
22.1.1. ***Brush Size*** : 34
22.1.2. ***Opacity*** : 100

23. Peignez autour du volcan pour créer une surface plus lisse. Faites attention de ne pas trop adoucir afin de garder une forme creuse.



Vous avez la forme de base de votre île, vous allez ajouter des textures pour produire un effet dans la scène.

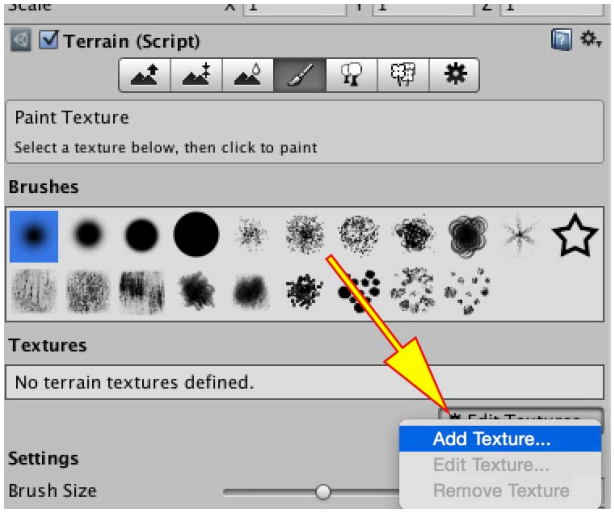
Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

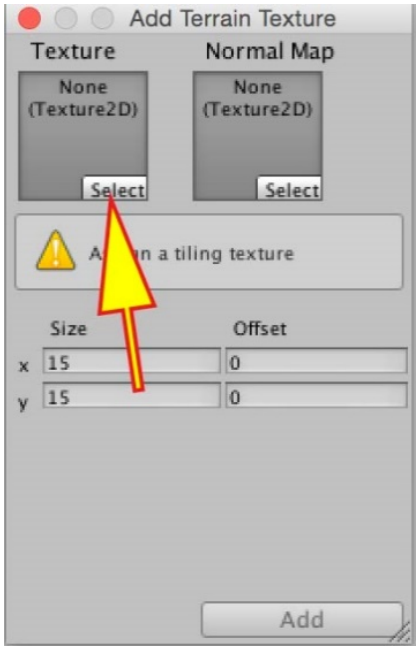
Ajouts de textures

La première chose à retenir lorsque vous texturez votre terrain, c'est que la première texture choisie sera colorée par défaut sur la totalité de la scène.

24. Cliquez sur la 4e icône de l'outil terrain pour ouvrir : ***Paint texture***.

25. Dans *l'Inspector,* appuyez sur le bouton « Edit Textures », puis « Add Texture… »



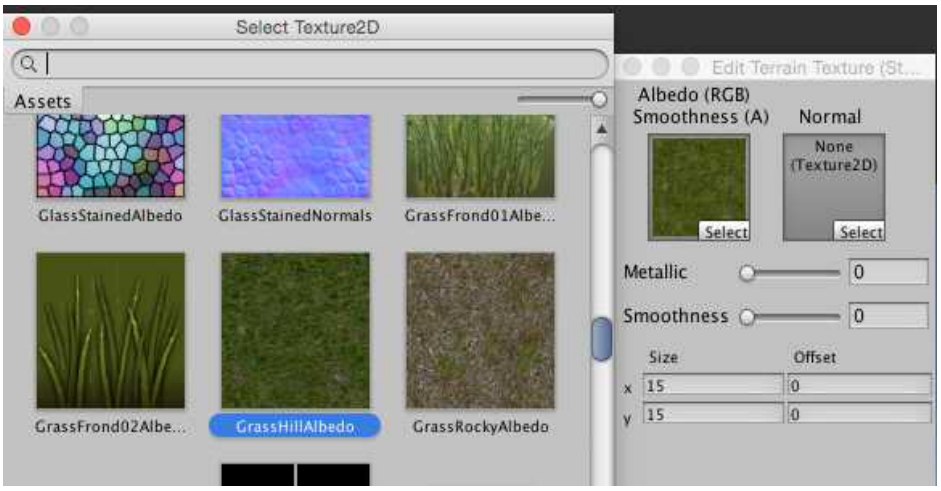


La fenêtre : *Add Terrain Texture* va s'ouvrir.

26. Appuyez sur le bouton « Select » en-dessous du premier espace pour les textures.

Une seconde fenêtre va s'ouvrir.

27. Dans cette fenêtre, recherchez la texture avec le nom **GrassHillAlbedo** et sélectionnez-la.

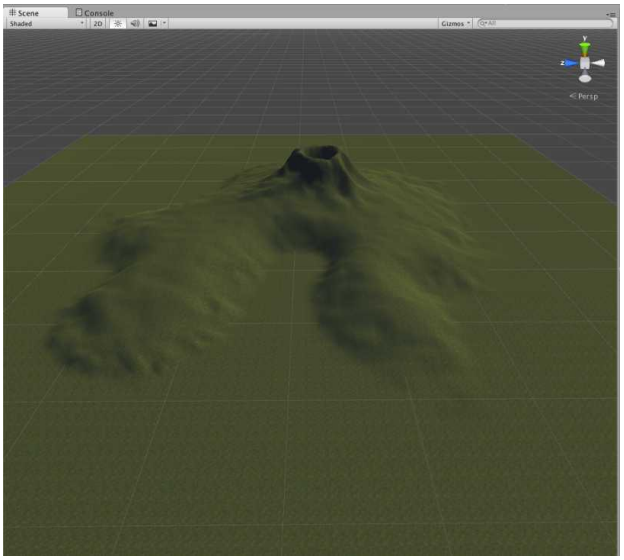


Un aperçu de la texture va apparaître dans la première fenêtre.

28. Assurez-vous que la taille : ***Size*** de la texture est **x : 15** et **y : 15**

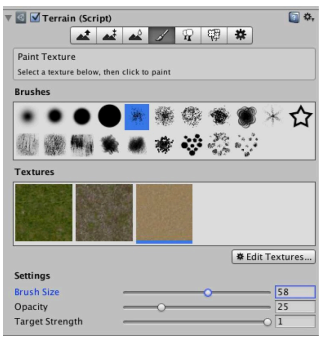
29. Appuyez sur le bouton : « Add » pour ajouter la nouvelle texture.

Vous devriez voir la texture de gazon sur toute la surface de la scène.

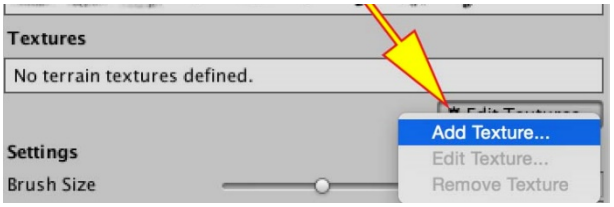


30. Refaites les mêmes étapes pour les textures :

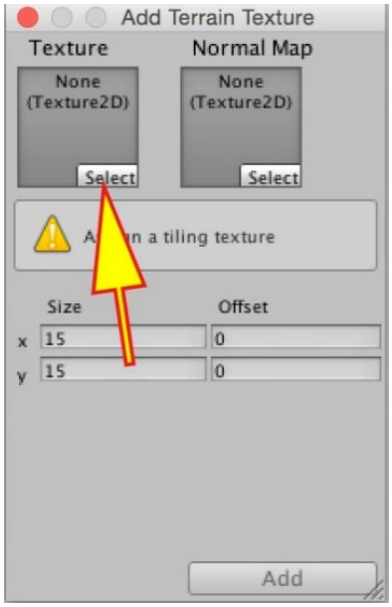
- **GrassRockyAlbedo**
- **SandAlbedo**



Certaines textures ont également un type : ***normal map*** pour ajouter du relief.



31.Dans *l'Inspector,* appuyez sur le bouton « Edit Textures », puis « Add Texture… »



La fenêtre *Add Terrain Texture* va s'ouvrir.

32. Appuyez sur le bouton « Select » en-dessous de la première case des textures.

Une seconde fenêtre va s'ouvrir.

33. Recherchez la texture avec le nom **MudRockyAlbedoSmoothness** et sélectionnez-la.

L'aperçu de la texture va apparaître dans la première icône.

34. Appuyez sur le bouton « Select » en-dessous du deuxième espace des textures.

Une seconde fenêtre va s'ouvrir.

35. Recherchez la texture **MudRockyNormal** et sélectionnez-la.

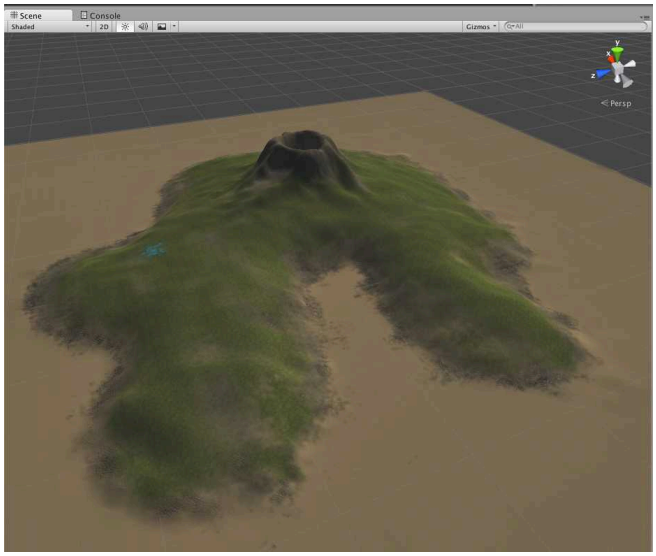
36. Assurez-vous que la taille : ***size*** de la texture est **x : 15** et **y : 15.**

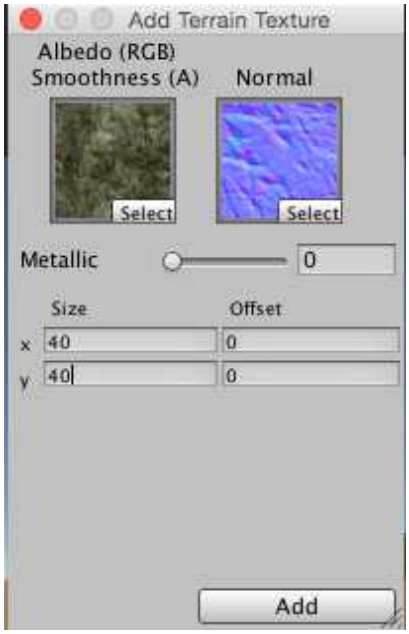
37. Appuyez sur le bouton « Add » pour ajouter la nouvelle taille à la liste texture.

37.1. Changez les paramètres suivants avant de peindre les textures :

37.1.1. ***Brushes*** : Particules ou flocons.
37.1.2. ***Brushes Size*** : entre 100 et 30 selon les besoins.
37.1.3. ***Opacity*** : entre 100 et 25 selon les besoins.
37.1.4. ***Target Strength*** : 1

38. Peignez les différentes textures sur le terrain.



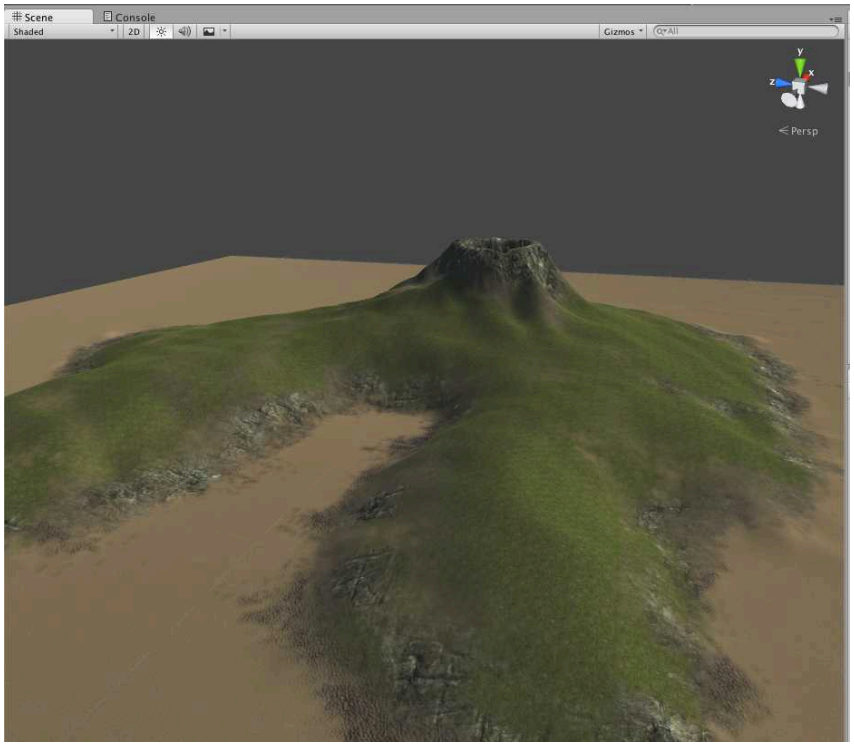


39.Ajoutez une dernière texture avec les images : **CliffAlbedoSpecular** et **CliffNormal.**

39.1. De plus, changez la taille : **size** de celle-ci pour **x: 40** et **y: 40**

40.Appuyez sur « Add » pour ajouter la texture.

41.Peignez cette texture sur les récifs et à l'intérieur du volcan.



Il y a encore quelques éléments à ajouter au terrain pour que celui-ci soit plus réaliste. Unity® offre plusieurs outils pour personnaliser les environnements, dont un éditeur d'arbres.

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Semons des arbres

1. Dans la barre de menus, cliquez sur :

   GameObject | 3D Object | Tree

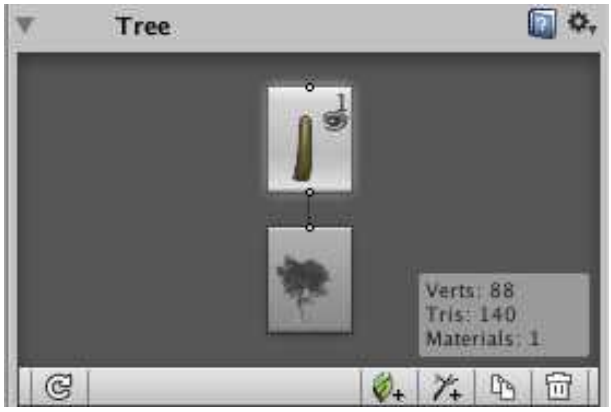
2. Sélectionnez l'objet **tree** dans votre *Hierarchy*.

3. Placez votre curseur au-dessus de la fenêtre *Scene* et appuyez sur la touche **(F)** de votre clavier pour focaliser le tronc de l'arbre.

Dans votre *Inspector*, vous devriez voir les outils de création d'arbres. Ces outils vous permettent de générer automatiquement des branches et des feuilles. Vous pouvez également dessiner manuellement les branches.

Pour cet exercice, vous allez créer automatiquement les différentes parties.

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.



Notez-bien!

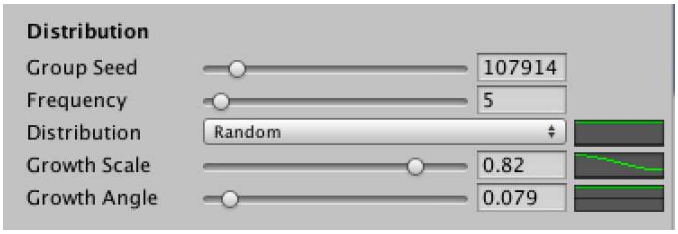
Nous reviendrons bientôt sur notre exercice. Pour le moment, j'attire votre attention juste en-dessous de la fenêtre d'arborescence, vous pouvez observer cinq icônes. Le premier, rafraîchit les données et les quatre autres sont plus importants.

Outils de création	
	Le premier bouton attache des feuilles à la branche sélectionnée.
	Le deuxième ajoute des branches au tronc (ou à la branche).
	Le troisième duplique les branches ou les feuilles sélectionnées.
	Le dernier bouton supprime les branches ou les feuilles choisies.

L'outil des branches

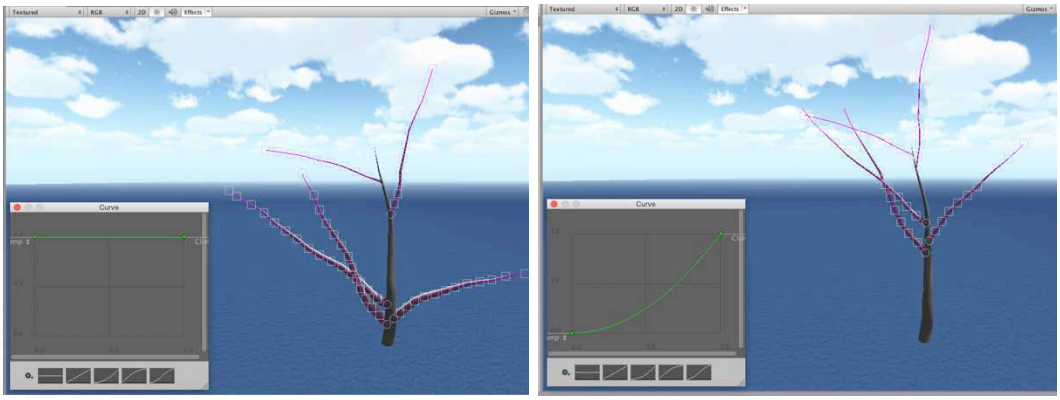
Distribution

Cette catégorie gère la quantité de branches qui seront attachées au tronc ainsi que l'endroit où elles seront placées sur celui-ci.



Distribution		
Group Seed	Nombre aléatoire qui dicte l'auto-génération des branches. En changeant ce nombre, les branches se disperseront différemment	
Frequency	Combien de branches sont ajoutées.	
Distribution	Dicte la méthode utilisée pour attacher les branches parmi quatre choix :	Aléatoire : ***Random***
		Alternance : ***Alternate***
		Opposé : ***Opposite***
		Tourbillonné : ***Whorled***
Growth Scale	Allonge ou raccourcit la longueur des branches. Plus le nombre est petit, plus les branches sont longues.	
Growth Angle	Dans quelle direction les branches poussent initialement. Plus le nombre est élevé, plus les branches poussent vers le haut de son parent.	

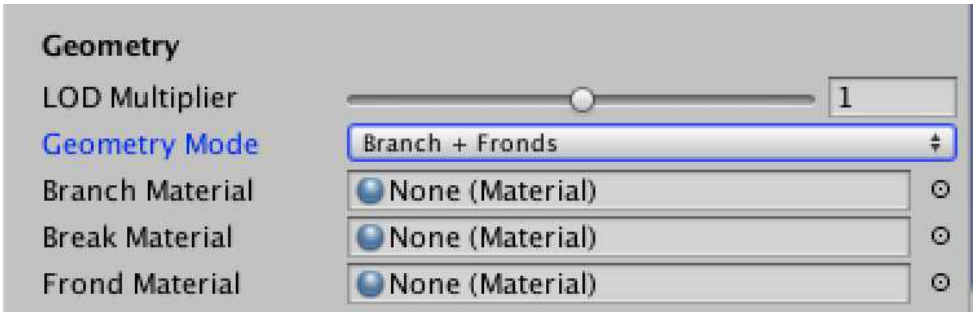
Les trois dernières options ont également des courbes pour contrôler ces paramètres.



Vous pouvez aussi utiliser l'outil de dessin main libre pour tracer vous-même vos branches manuellement.

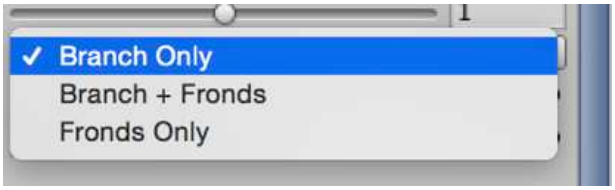
Maintenant, il faut aborder les paramètres de la section : ***Geometry***.

Geometry



Geometry	
LOD Multiplier	Ajuste la qualité du groupe sélectionné, relativement à la qualité globale du niveau des détails (LOD) de l'arbre. Ceci permet de prioriser les parties de l'arbre que l'on veut garder détaillées, au loin.
Geometry Mode	Permet de choisir si on travaille uniquement la branche : **Branch Only**
	la branche avec des feuilles lattérales : **Branch+Fronds**
	les feuilles lattérales uniquement : **fronds Only** (utile pour créer des fougères)
Branch Material	***Material*** principal pour les branches
Break Material	***Material*** pour le nœud des branches brisées

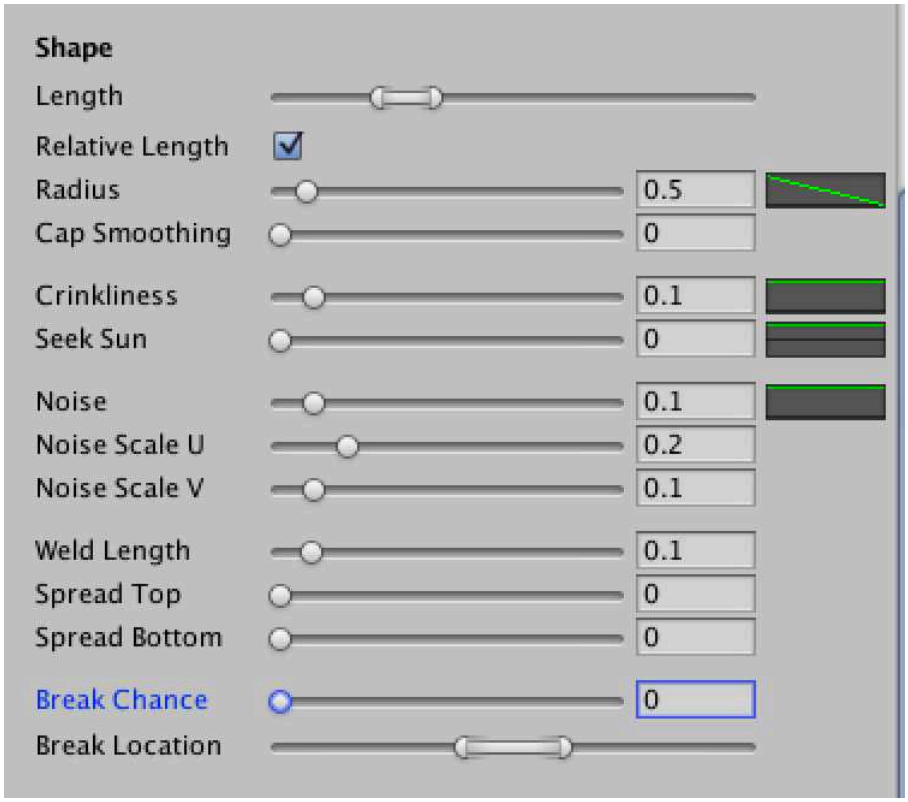
Geometry	
Frond Material	***Materiel*** pour les feuilles lattérales



Passez maintenant à la section suivante : ***Shape.***

Shape

Cette catégorie permet d'ajuster la forme et la taille des branches. Les courbes peuvent aider à ajuster les détails.

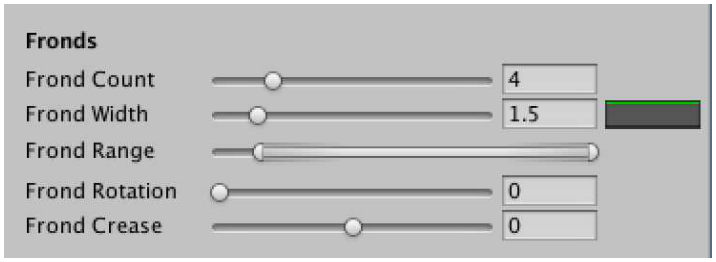


Shape	
Length	Ajuste la longueur des branches.
Relative Length	Détermine si la circonférence de la branche est relative à sa longueur.
Radius	Détermine la circonférence de la branche. La courbe permet d'ajuster comment celle-ci fluctue le long de la tige.
Cap Smoothing	Détermine la rondeur du bout de la branche. (Utile pour faire des cactus).
Crinkliness	Ajuste comment la branche est tordue. La courbe permet d'ajuster les fins détails.
Seek Sun	Change la façon dont les branches poussent (vers le haut ou le bas).
Noise	Augmente la complexité aléatoire de la surface.
Noise Scale U	Ajuste la complexité le long de la branche.
Noise Scale V	Ajuste la complexité le long de la branche. (Angle différent)
Weld Lenght	Change la taille de la jonction de la tige.
Spread Top	Facteur de répartition des jonctions dans le haut de la branche, relativement à son parent.
Spread Bottom	Facteur de répartition des jonctions dans le bas de la branche, relativement à son parent.
Break Chance	Probabilité qu'une branche soit brisée.
Break Location	Définit une zone où la branche peut se briser.

La prochaine section est seulement accessible quand la propriété : ***Fronds*** est activée.

Fronds

Cette catégorie est disponible si ***Geometry mode*** utilise les ***Fronds.*** Ceci sert à ajuster le nombre de feuilles lattérales ainsi que leurs propriétés.



Fronds	
Frond Count	Définit le nombre de feuilles autour de la branche.
Frond Width	Définit la longueur des feuilles.
Frond Range	Ajuste la position où les feuilles débutent et se terminent.
Frond Rotation	Définit la rotation des feuilles autour de la branche.
Frond Crease	Change la façon dont les feuilles se contractent le long de la tige.

Pour conclure, la dernière section sert à contrôler le vent dans les branches : ***Wind***

Wind

C'est ici que l'on ajuste l'animation des branches. Les zones de vent sont utilisées pour définir l'intensité des animations.

Notez-bien!

*Les zones de vent (**Wind Zones**) sont actives seulement durant le déroulement du jeu.*

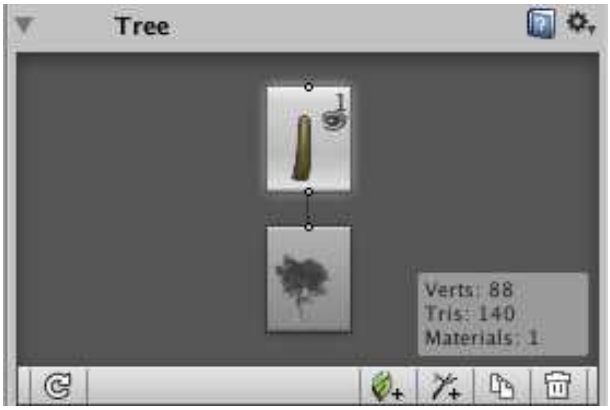
Wind	
Main Wind	Effet principal, gère le mouvement des branches et des feuilles de façon fluide.
Main Turbulence	Turbulence des branches et des feuilles globalement
Edge Turbulence	Turbulence des feuilles dans leurs extrémités
Create Wind Zone	Crée une zone de vent.

De retour à l'exercice

> **Conseil!**
>
> Sentez-vous à l'aise d'utiliser des valeurs différentes de celles proposées pour vos arbres.

4. Utilisez l'objet **tree** sélectionné.

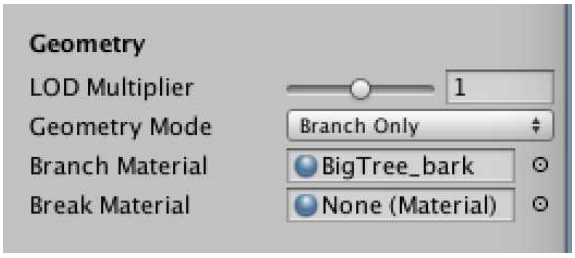
Dans *l'Inspector,* il y a un sous-menu : ***Tree.*** Vous devriez voir 2 images : L'image du bas gère les paramètres globaux et l'image du haut gère les propriétés du tronc ou des branches.



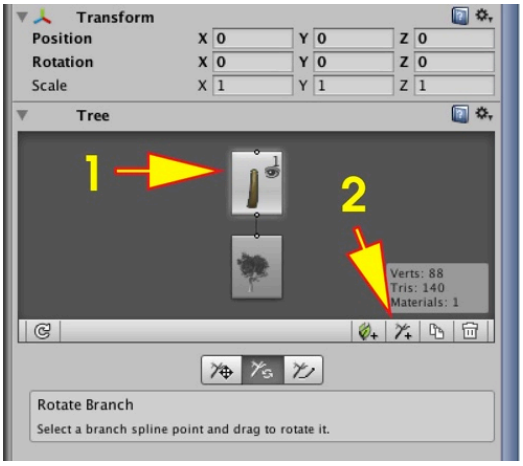
5. Appuyez sur l'image du tronc (image du haut) pour le mettre en surbrillance.

Avant d'ajouter des branches, vous allez mettre un ***Material*** sur le tronc.

6. Dans *l'Inspector*, localisez la catégorie ***Geometry.***

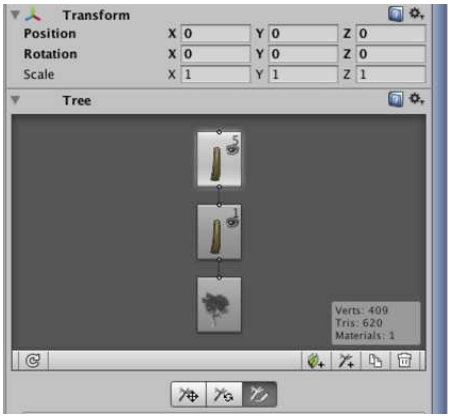


6.1. Changez uniquement ***Branch Material*** pour : **BigTree_bark**



Maintenant, le tronc n'a plus l'air d'être mort, enchaînez en attachant quelques branches à celui-ci.

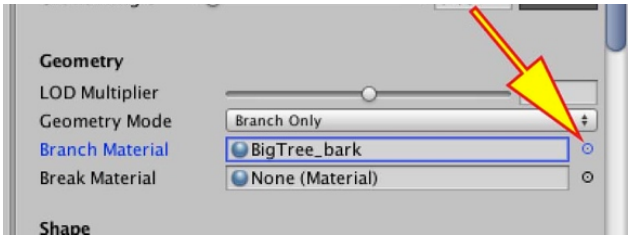
7. Appuyez sur le bouton ajout de branches « Add Branch Group ».



Maintenant que vous avez une branche d'attachée au tronc, vous allez faire un parcours d'ensemble des paramètres disponibles pour les branches.

7.1. Dans *l'Inspector*, changez les paramètres suivants :

Distribution
7.1.1. ***Group Seed*** : 107914
7.1.2. ***Frequency*** : 5
7.1.3. ***Distribution*** : Random, (Courbe) : horizontale
7.1.4. ***Growth Scale*** : 0.82, (Courbe) : décroissante-sinus
7.1.5. ***Growth Angle*** : 0.079, (Courbe) : horizontale

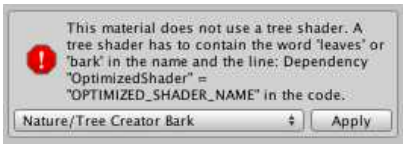


Geometry
7.1.6. ***LOD Multiplier*** : 1.0
7.1.7. ***Geometry Mode*** : Branch Only
7.1.8. ***Branch Material*** **:** Branches_0
7.1.9. ***Break Material*** **:** Aucun

Shape
7.1.10. ***Length*** **:** 12%-20%
7.1.11. ***Relative Length*** **:** Oui
7.1.12. ***Radius*** **:** 0.8, (Courbe) : décroissante-linéaire
7.1.13. ***Cap Smoothing*** **:** 0
7.1.14. ***Crinkliness*** **:** 0.136, (Courbe) : Horizontale
7.1.15. ***Seek Sun*** **:** 0.216, (Courbe) : Horizontale
7.1.16. ***Noise*** **:** 0.1, (Courbe) : Horizontale
7.1.17. ***Noise Scale*** **U :** 0.2
7.1.18. ***Noise Scale*** **V :** 0.1
7.1.19. ***Weld Length*** **:** 0.1
7.1.20. ***Spread Top*** **:** 0
7.1.21. ***Spead Bottom*** **:** 0
7.1.22. ***Break Chance*** **:** 0
7.1.23. ***Break Location*** **:** 45%-55%

Wind
7.1.24. ***Main Wind*** **:** 0.5
7.1.25. ***Main Turbulence*** **:** 0.5

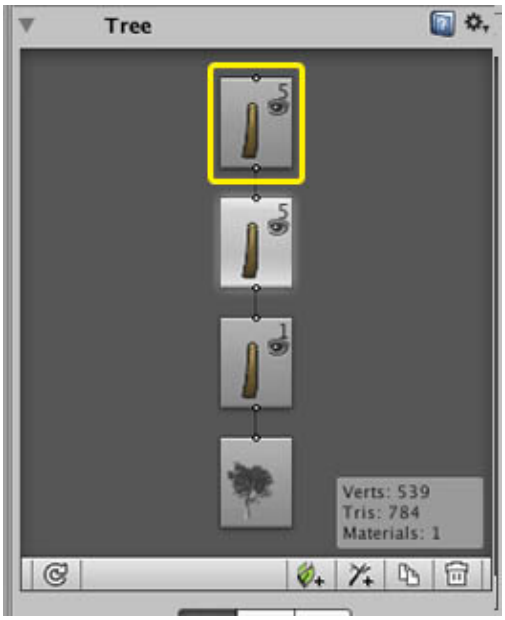
Vous verrez ce message, la première fois que vous appliquerez le ***Material*** : **Branche_0**.



7.2. Appuyez simplement sur le bouton « Apply » pour fixer ce problème.

Vous allez attacher quelques petites branches additionnelles aux grandes branches de l'étape précédente.

8. Appuyez sur le bouton ajout de branches « Add Branch Group ».

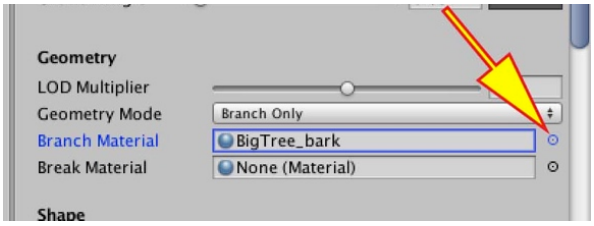


En détails...

Comme vous pouvez le constater, les nouvelles branches sont enfants des branches précédentes qui, elles-mêmes, sont enfants du tronc.

8.1. Dans *l'Inspector*, entrez les paramètres suivants :

Distribution
8.1.1. ***Group Seed*** : 1234
8.1.2. ***Frequency*** : 1
8.1.3. ***Distribution*** : Random, (Courbe) : horizontale
8.1.4. ***Growth Scale*** : 1, (Courbe) : décroissante-sinus
8.1.5. ***Growth Angle*** : 0.079, (Courbe) : horizontale



Geometry
8.1.6. ***LOD Multiplier*** : 1.0
8.1.7. ***Geometry Mode*** : Branch Only.
8.1.8. ***Branch Material*** **:** Branches_0
8.1.9. ***Break Material*** **:** Aucun

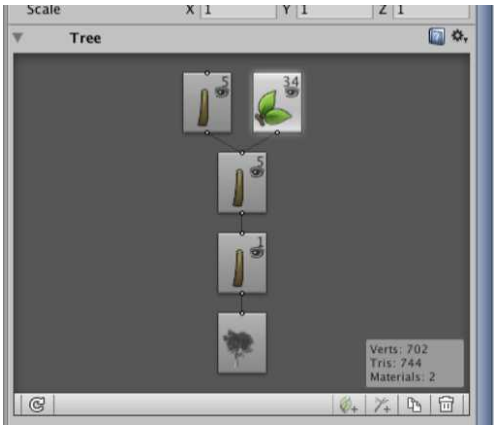
Shape
8.1.10. ***Length*** **:** 12%-20%
8.1.11. ***Relative Length*** **:** Oui
8.1.12. ***Radius*** **:** 0.5, (Courbe) : décroissante-linéaire
8.1.13. ***Cap Smoothing*** **:** 0
8.1.14. ***Crinkliness*** **:** 0.093, (Courbe) : Horizontale
8.1.15. ***Seek Sun*** **:** 0.374, (Courbe) : Horizontale

8.1.16. ***Noise*** **:** 0.1, (Courbe) : Horizontale
8.1.17. ***Noise Scale*** **U :** 0.2
8.1.18. ***Noise Scale*** **V :** 0.1
8.1.19. ***Weld Length*** **:** 0.1
8.1.20. ***Spread Top*** **:** 0
8.1.21. ***Spead Bottom*** **:** 0
8.1.22. ***Break Chance*** **:** 0
8.1.23. ***Break Location*** **:** 45%-55%

Wind
8.1.24. ***Main Wind*** **:** 0.5
8.1.25. ***Main Turbulence*** **:** 0.5

Vous allez attacher quelques feuilles au bout des deux derniers niveaux de branches.

9. Sélectionnez le 2e niveau de branches (au centre, juste après le tronc) dans l'arborescence.



9.1. Appuyez sur le bouton : « Add Leaf Group » pour ajouter des feuilles sur ces branches.

Après avoir ajouté du feuillage, vous allez faire le tour des options disponibles.

L'outil de feuillage

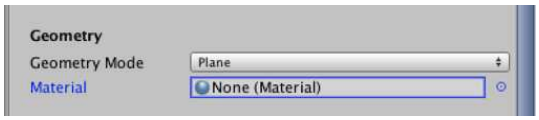
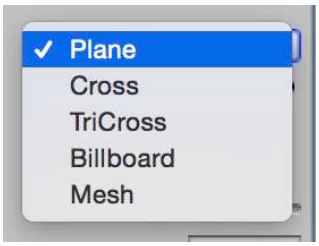
Distribution

Cette catégorie gère la quantité des feuilles qui seront attachées à la branche ainsi que l'endroit où elles seront placées sur celle-ci.

Comme les options sont les mêmes que pour les branches, vous allez passer directement à la section ***Geometry***.

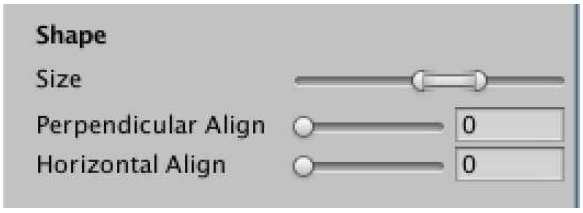
Geometry

Cette catégorie gère la façon dont les feuilles sont affichées sur les branches.

Geometry		
Geometry Mode	 Définit quel type de géométrie est utilisé pour la disposition du feuillage 	Plat : ***Plane***
		Croix : ***Cros***
		Triple croix : ***TriCross***
		Plan fixe : ***Billboard***
		Modèle 3D : ***Mesh***
Material	Est utilisé pour le feuillage.	
Mesh		Permet de choisir un modèle 3D personnalisé pour les feuilles, lorsque l'option ***Mesh*** est utilisée pour ***Geometry Mode***.

Shape

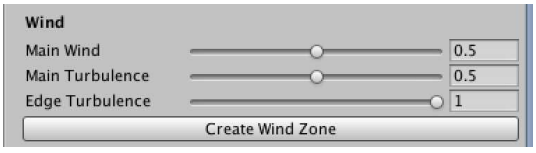
Ajuste la forme et la taille des feuilles.



Shape	
Size	Ajuste la taille des feuilles. Il utilise les extrémités pour définir le ratio des différentes tailles du feuillage.
Perpendicular Align	Ajuste l'angle des feuilles perpendiculairement à sa branche.
Horizontal Align	Ajuste l'angle par rapport à l'horizon.

Wind

Ajuste le vent, (les paramètres sont les mêmes que pour les branches).



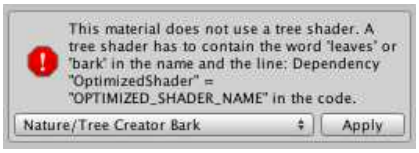
De retour à l'exercice

> **Conseil!**
>
> Nous venons de parcourir les différents paramètres pour la création du feuillage. Les paramètres suivants ne sont que des suggestions. Sentez-vous à l'aise d'explorer vos propres valeurs.

10. Pour cet exercice, utilisez les paramètres suivants :

Distribution
10.0.1. ***Group Seed*** : 1234
10.0.2. ***Frequency*** : 8
10.0.3. ***Distribution*** : Random, (Courbe) : horizontale
10.0.4. ***Growth Scale*** : 1, (Courbe) : décroissante-sinus
10.0.5. ***Growth Angle*** : 0, (Courbe) : horizontale

Geometry
10.0.6. ***Geometry Mode*** **:** Plane
10.0.7. ***Material*** **:** Leaves_1



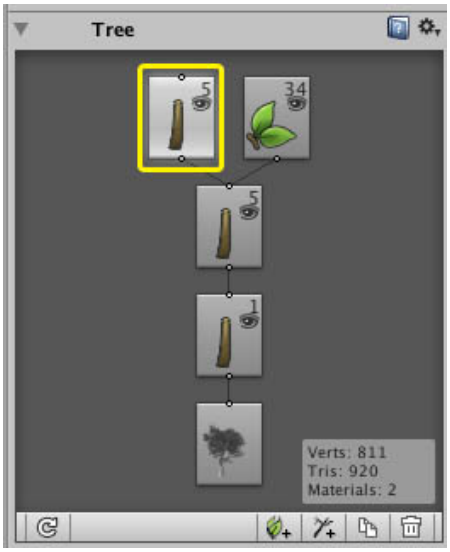
Il est possible de voir ce message la première fois que vous appliquez cette fonction ***Material*** :

10.1. Appuyez simplement sur le bouton « Apply » pour rapidement résoudre ce problème.

Shape
10.1.1. ***Size*** **:** 55%-70%
10.1.2. ***Perpendicular Align*** **:** 0
10.1.3. ***Horizontal Align*** **:** 0

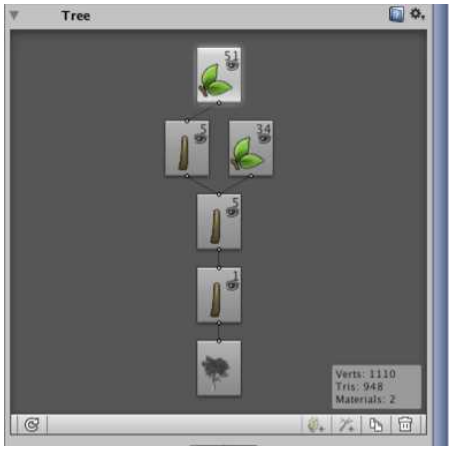
Wind
10.1.4. ***Main Wind*** **:** 0.5
10.1.5. ***Main Turbulence*** **:** 0.5
10.1.6. ***Edge Turbulence*** **:** 1

Sélectionnez la dernière branche au bout de l'arborescence.



11. Appuyez sur le bouton ajout de feuilles à la branche « Add Leaf Group ».

11.1. Utilisez les paramètres suivants :

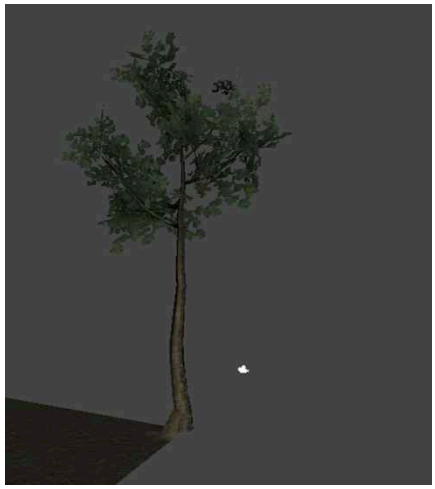


Distribution
11.1.1. ***Group Seed*** : 1234
11.1.2. ***Frequency*** : 17
11.1.3. ***Distribution*** : Random, (Courbe) : horizontale
11.1.4. ***Growth Scale*** : 1, (Courbe) : décroissante-sinus
11.1.5. ***Growth Angle*** : 0, (Courbe) : horizontale

Geometry
11.1.6. ***Geometry Mode*** **:** Plane
11.1.7. ***Material*** **:** Leaves_1

Shape
11.1.8. ***Size*** **:** 55%-70%
11.1.9. ***Perpendicular Align*** **:** 0
11.1.10. ***Horizontal Align*** **:** 0

Wind
11.1.11. ***Main Wind*** **:** 0.5
11.1.12. ***Main Turbulence*** **:** 0.5
11.1.13. ***Edge Turbulence*** **:** 1



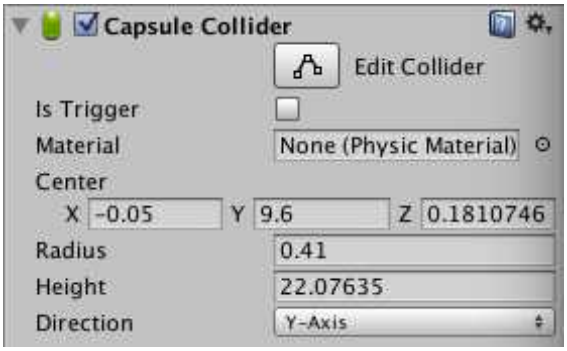
Voici les résultats obtenus avec ces paramètres :

Vous allez attacher une ***capsule collider*** à cet arbre afin que les collisions fonctionnent.

12. Sélectionnez votre arbre.

12.1. Appuyez sur le bouton « Add Component », puis :

Physics | Capsule Collider



12.2.Ajustez les valeurs des paramètres : (***Center, Radius*** et ***Height***) pour qu'ils couvrent correctement votre tronc.

Vous avez maintenant un modèle d'arbre. Vous allez utiliser ***Prefab*** qui est généré automatiquement dans *Project* afin de le rendre réutilisable et facilement modifiable.

13. Dans le panneau : *Project*, recherchez l'objet qui porte le même nom que votre arbre dans la scène : (ex : **Tree, Tree 1**…).

13.1. Renommez ce ***Prefab*** : Tree_prefab.

14. Une fois le ***Prefab*** nommé, supprimez le modèle de la scène.

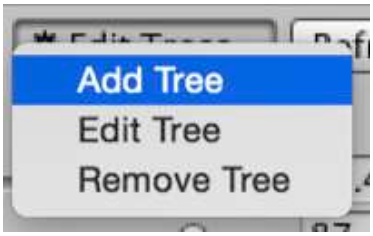
15. Répétez les étapes précédentes pour faire quelques variations d'arbres.

Vous allez maintenant retourner à votre scène afin de mettre des arbres sur le terrain.

16. Sélectionnez le terrain dans *Hierarchy.*

17. Dans *l'Inspector et* dans les outils du terrain, cliquez sur la 5e icône pour placer des arbres : ***Place Tree***.

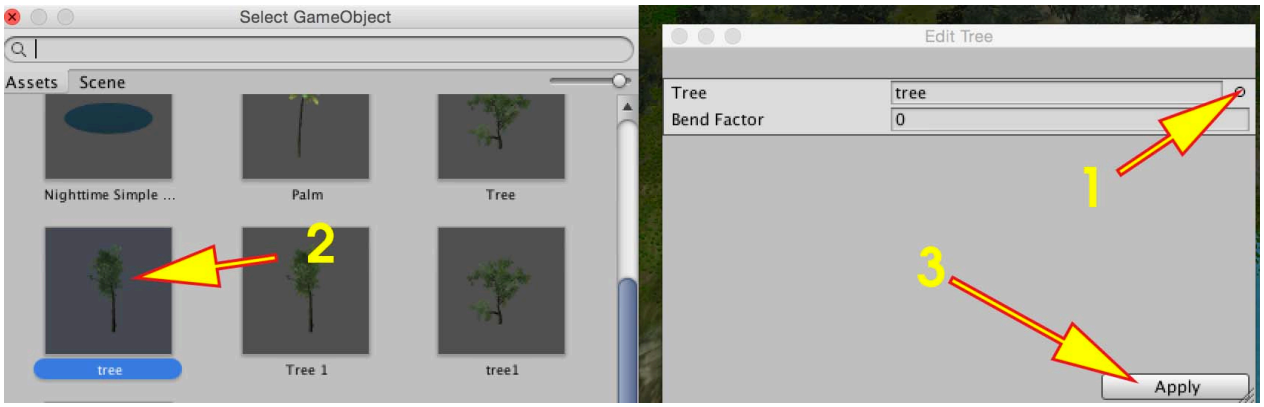
18. Appuyez sur le bouton « Edit Trees » et puis « Add Tree ».



La fenêtre *Edit Tree* apparaît.

19. Ajoutez votre ***Prefab*** dans la case ***Tree.***

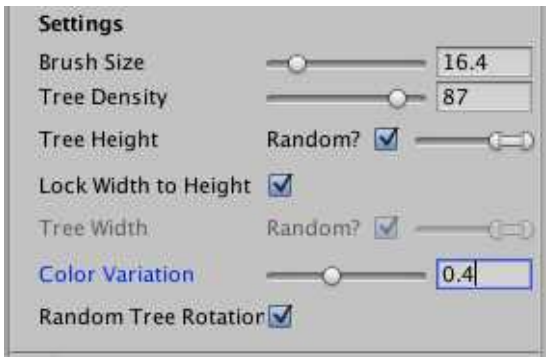
19.1. Appuyez sur « Apply ».



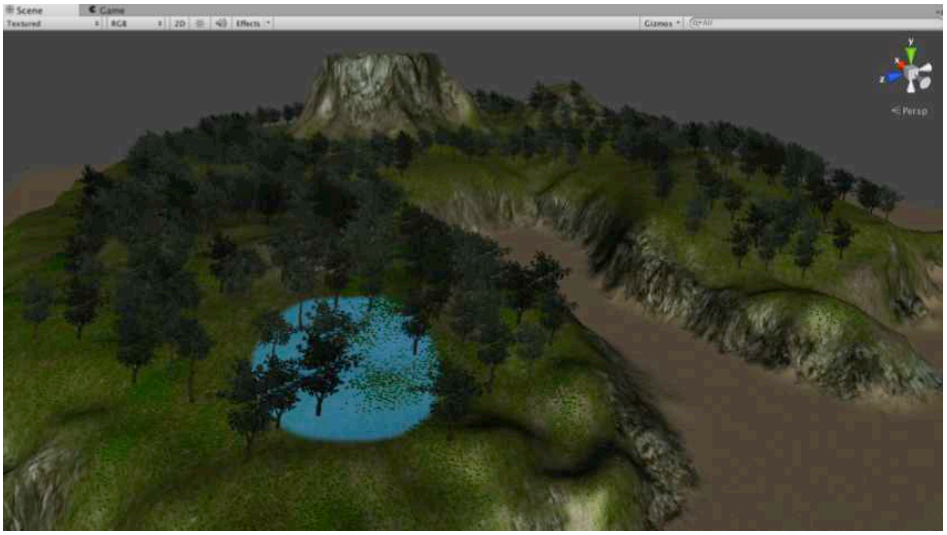
Maintenant que vous avez un modèle en mémoire, vous allez peindre de petites forêts avec une brosse.

20. Changez les paramètres ***Settings*** suivants :

Settings
20.0.1. ***Brush Size*** : 16.4
20.0.2. ***Tree Density*** : 87
20.0.3. ***Tree Height*** : Random?: oui, pour le dernier 10%
20.0.4. ***Lock Width to Height***: oui
20.0.5. ***Color Variation***: 0.4
20.0.6. ***Random Tree Rotation***: oui



21. Peignez votre forêt sur l'île.



Il y a maintenant de la végétation dans la scène, il ne manque que des brins de gazon.

Ce type de végétation est seulement visible à une certaine distance et *Unity*® restreint l'affichage du gazon dans l'éditeur. C'est la raison pour laquelle, avant de poursuivre, vous allez retourner dans les ***paramètres*** du terrain.

22. Cliquez sur la dernière icône des outils de l'éditeur de terrain : ***terrain settings.***

Jetez un coup d'œil sur les différentes options survolées jusqu'à présent.

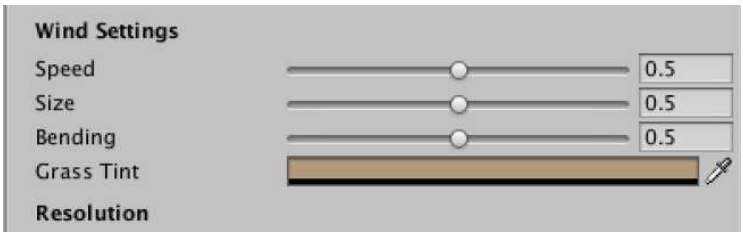
Les paramètres du terrain

Tree & Detail Objects

Tree & Detail Objects	
Draw	Indique que l'on devrait afficher la végétation.
Bake Light Probes For Trees	Permet de calculer les ***light probes*** pour les arbres.
Detail Distance	Indique à quelle distance de la caméra doit arrêter l'affichage de détails (gazon).
Collect Detail Patches	Option qui permet de charger les éléments de ***details*** en mémoire, afin d'améliorer les performances au détriment d'un peu de mémoire.
Detail Density	Indique la densité de détails à la fois (aide à améliorer les performances).
Tree Distance	Indique à quelle distance de la caméra il faut arrêter l'affichage des arbres.
Billboard Start	Indique à quelle distance de la caméra, le modèle 3D de l'arbre est remplacé par un plan 2D : ***billboard.***
Fade Length	Donne la distance de la transition entre le plan 2D et le modèle 3D.

Tree & Detail Objects	
Max Mesh Trees	Indique le nombre d'arbres visibles qui sont représentés par des modèles 3D. Les autres modèles seront remplacés par des plans 2D.

Wind Settings



Wind Settings	
Speed	La vitesse du vent sur le gazon.
Size	La taille des bouffées de vent.
Bending	Degré d'inclinaison du gazon dans le vent.
Grass Tint	Teinte appliquée sur l'ensemble du gazon.

Pour les besoins de cet exercice, vous allez changer la distance d'affichage des détails afin de mieux voir le gazon.

23. Changez la valeur des paramètres suivants :

Tree & Detail Objects
23.0.1. ***Detail Distance*** **:** 250
23.0.2. ***Detail Density*** **:** 1
23.0.3. ***Tree Distance*** **:** 2000
23.0.4. ***Billboard Start*** **:** 300
23.0.5. ***Fade Length*** **:** 200
23.0.6. ***Max Mesh Trees*** **:** 150

Notez-bien!

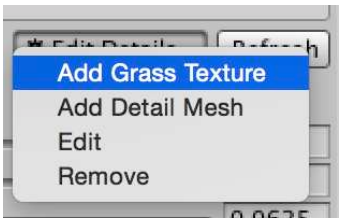
Il est important d'augmenter la distance d'affichage des détails, sinon il sera difficile de voir le gazon pendant sa création.

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

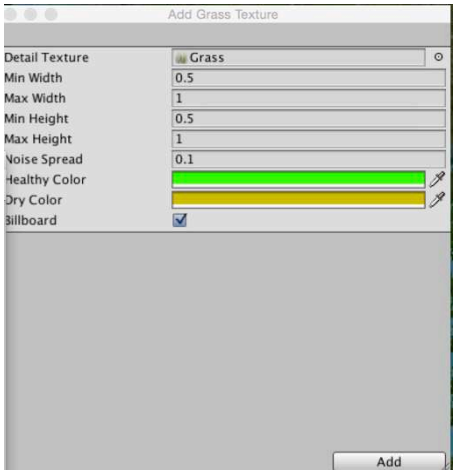
Maintenant, vous pouvez appliquer les détails dans la scène.

24. Dans l'*Inspector*, cliquez sur la 6e icône des outils de terrain : ***Paint Details***.

25. Cliquez sur le bouton « Edit Details… », puis « Add Grass Texture ».



Add Grass Texture

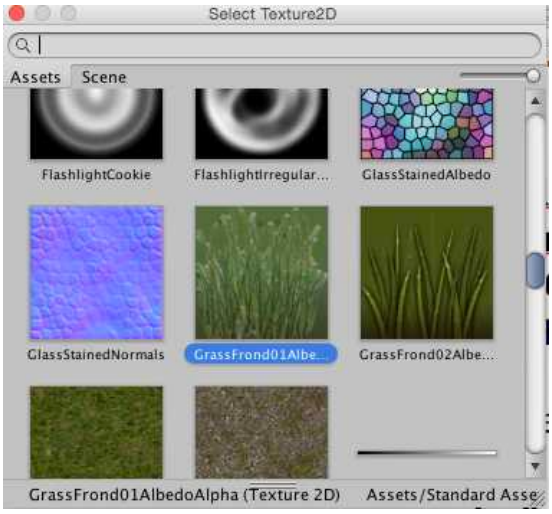


La fenêtre *Add Grass Texture* apparaît.

Cette fenêtre contrôle la taille et la couleur du gazon.

Voici les différentes options disponibles :

Add-Grass Texture	
Detail Texture	La texture utilisée pour le gazon.
Min Width	Largeur minimale du gazon.
Max Width	Largeur maximale du gazon.
Min Height	Hauteur minimale du gazon.
Max Height	Hauteur maximale du gazon.
Noise Spread	Contrôle le nombre de variations de gazon dans une zone.
Healthy Color	Couleur du gazon, quand il est frais.
Dry Color	Couleur du gazon, quand il est sec.
Billboard	Est-ce que le plan 2D doit suivre l'orientation de la caméra ?



26. Pour cet exercice, utilisez les paramètres suivants :

26.0.1. ***Detail Texture***: GrassFrond01AlbedoAlpha
26.0.2. ***Min Width***: 0.5
26.0.3. ***Max Width***: 1
26.0.4. ***Min Height***: 0.5
26.0.5. ***Max Height***: 1
26.0.6. ***Noice Spread*** : 0.1
26.0.7. ***Healthy Color***: Beau vert frais
26.0.8. ***Dry Color***: Jaune gazon sec
26.0.9. ***Billboard***: oui

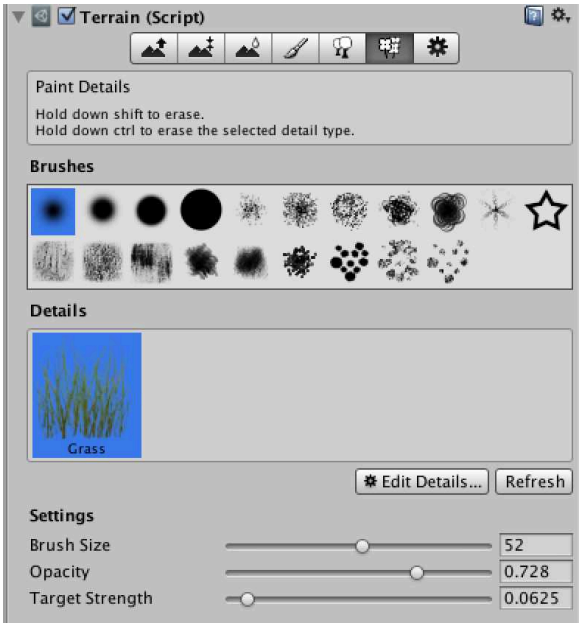
27. Appuyez sur le bouton : « Apply ».

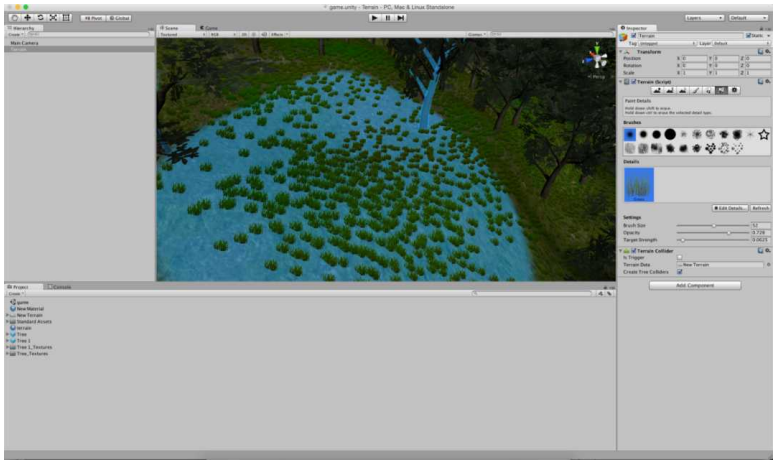
Ajoutez du gazon sur le terrain, mais auparavant :

28. Changez les paramètres suivants :

28.0.1. ***Brush Size*** : 52
28.0.2. ***Opacity*** : 0.73
28.0.3. ***Target Strength*** : 0.0625

29. Au besoin, approchez la caméra dans la scène et ajoutez des zones de gazon sur votre terrain.





Vous avez complété le terrain. Donnez-vous une tape dans le dos, puis préparez-vous pour les prochaines étapes comprenant la création de l'océan, le ciel, l'atmosphère et le soleil. Ne vous en faites pas, avec Unity, ce sera un jeu d'enfant.

Une goutte d'eau dans l'océan

Vous allez maintenant ajouter une étendue d'eau autour de votre île. Depuis *Unity*® 5, l'éditeur vient avec des modules préfabriqués pour créer des étendues d'eau plus réalistes qu'auparavant.

Notez-bien!

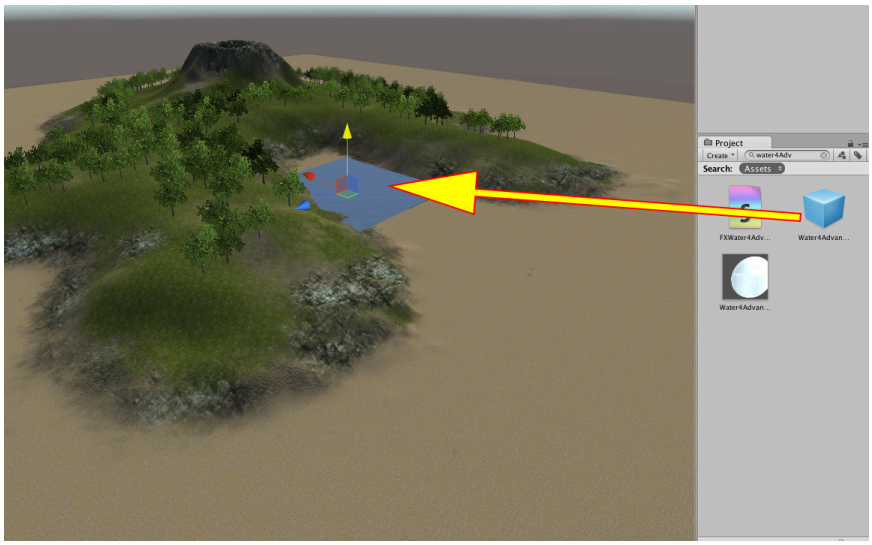
*Autrefois, uniquement le module **Water Basic** était fourni avec la version gratuite du logiciel afin de simuler des vagues sur une surface, mais ce module était opaque et visible sur un seul angle, ce qui manquait de réalisme.*

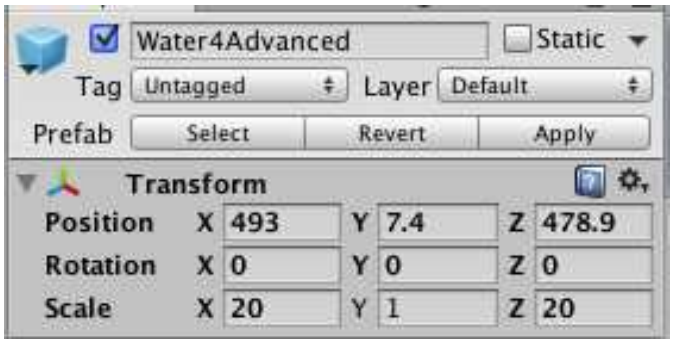
1. Dans le panneau : *Project*, recherchez un objet qui se nomme : **Water4Advanced**. Il se trouve dans le répertoire :

Standard Assets | Environment | Water | Water4 | Prefabs

2. Glissez **Water4Advanced** au centre de votre scène.

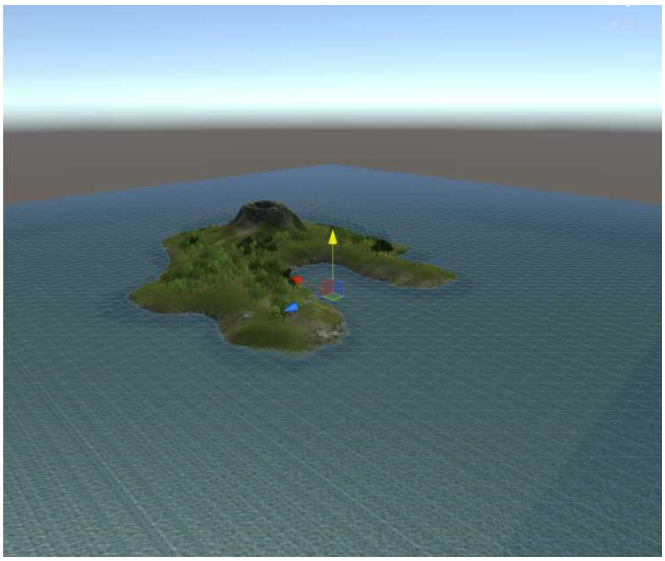
2.1. Changez les paramètres suivants pour ajuster la taille et la position de l'océan.





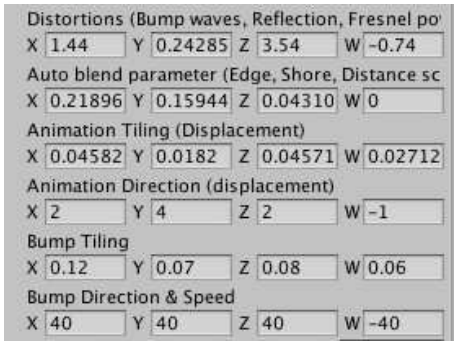
Transform
2.1.1. ***Position*** **: Y**= 7,4
2.1.2. ***Scale*** : **X**= 20, **Y=** 1, **Z**= 20

Voici le résultat escompté :



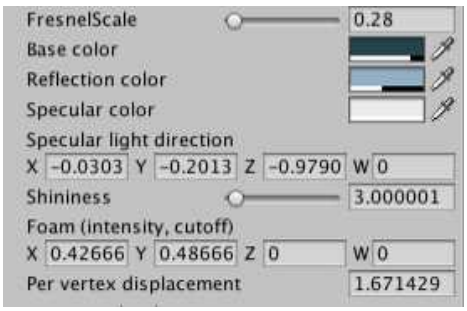
3. Pour rendre les vagues plus réalistes, recherchez le ***Material*** : **Water4Advanced**.

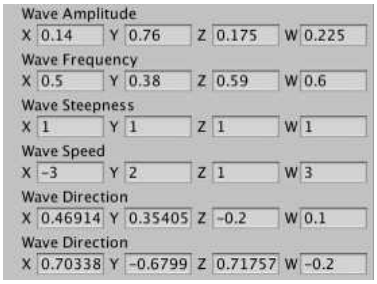
3.1. Dans *l'Inspector*, changez les paramètres suivants :



3.1.1. ***Distortions*** : **X=**1.44, **Y=**0.243, **Z=**3.54, **W=**-0.74
3.1.2. ***Auto blend parameter*** **: X=**0.219, **Y=**0.16, **Z=**0.043, **W=**0
3.1.3. ***Animation Tiling*** **: X=**0.046, **Y=**0.0182, **Z=**0.046, **W=**0.027
3.1.4. ***Animation Direction*** **: X=**2, **Y=**4, **Z=**2, **W=**-1
3.1.5. ***Bump Direction & Speed*** **: X=**40, **Y=**40, **Z=**40, **W=**-40

3.1.6. ***FresnelScale*** : 0.28
3.1.7. ***Base color*** **:** Bleu opaque**,** ***Alpha*** **:** 209
3.1.8. ***Reflection color*** **:** Bleu pâle**,** ***Alpha*** **:** 110
3.1.9. ***Specular color*** **:** Gris**,** ***Alpha*** **:** 255
3.1.10. ***Specular light direction*** **: X=**-0.03**, Y=**-0.2, **Z=**-0.98, **W=**0
3.1.11. ***Shininess*****:** 3.0
3.1.12. ***Foam*****: X=**0.426, **Y=**0.486, **Z=**0, **W=**0
3.1.13. ***Per vertex displacement*****:** 1.67
3.1.14. ***Wave Amplitude*** **: X=**0.14**, Y=**0.76**, Z=**0.175**, W=**0.225
3.1.15. ***Wave Frequency*** **: X=**0.5**, Y=**0.38**, Z=**0.59**, W=**0.6





3.1.16. ***Wave Steepness*** **: X=**1, **Y=**1, **Z=**1, **W=**1
3.1.17. ***Wave Speed*** **: X=**-3, **Y=**2, **Z=**1, **W=**3
3.1.18. ***Wave Direction*** **: X=**0.47, **Y=**0.354, **Z=**-0.2, **W=**0.1
3.1.19. ***Wave Direction*** **(2) : X=**0.7, **Y=**-0.68, **Z=**0.72, **W=**-0.2

Maintenant, ajouter un ***skybox*** pour remplacer le dégradé qui sert de ciel, par défaut.

Ajout d'un ciel et du brouillard dans la scène

Notez-bien!

Dans les anciennes versions d'Unity®, un ***package*** *nommé* ***Skyboxes*** *était fourni avec le logiciel. Par contre, Unity® 5 a remplacé plusieurs packages qui étaient inclus dans les versions antérieures. Certains paquets ont été mis à jour ou regroupés, mais les* ***skyboxes*** *ont tout simplement disparu.*

Depuis l'écriture de cet ouvrage, un nouveau paquet de ***skyboxes*** a peut-être fait son apparition. Dans la version 5.0, il faut se retourner vers l'***Assets Store.***

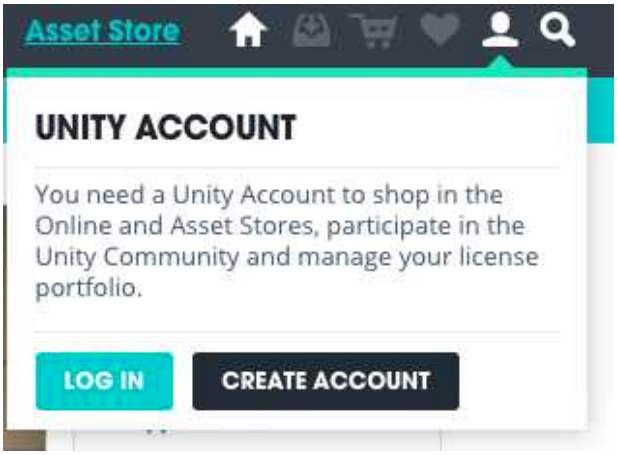
Si vous n'avez pas encore créé votre compte utilisateur Unity, vous devriez le faire avant de continuer cet exercice.

https://www.assetstore.unity3d.com

• **À partir de l'*Asset Store,* recherchez le bouton «** Create Account **» sur la page.**

↘ Il est possible que l'apparence du site soit différente de cette image.

• **Suivez les instructions pour créer votre compte.**

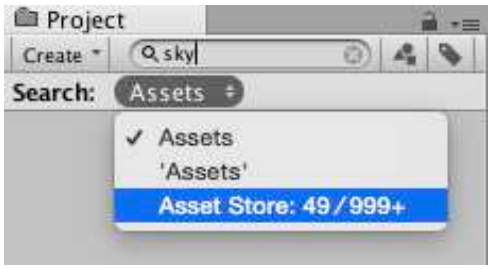


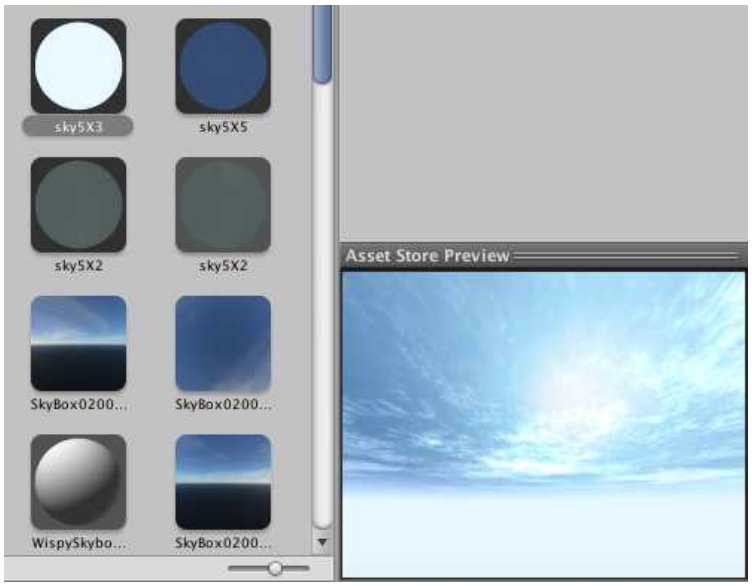
Notez-bien!

Vous pouvez également le faire lors de votre premier téléchargement.

4. Dans le panneau *Project*, faites une recherche : **sky** dans le champ de recherches.

4.1. Puis, changez l'emplacement de la recherche en cliquant sur le lien ***Assets*** et en le remplaçant par ***Asset Store***.





Une liste de ***skyboxes*** les plus populaires qui va être affichée. Les éléments gratuits sont classés séparément des payants.

En sélectionnant un élément dans la liste, vous verrez un aperçu (interactif) dans *l'Inspector*: ***Asset Store Preview.***

Lorsque que vous avez trouvé un paquet intéressant!

5. Appuyez sur le bouton « Import package » ou « Buy for » afin acquérir ce paquet.

Le *package* va se télécharger et se décompresser automatiquement.

6. Appuyez sur le bouton : « Import » pour ajouter les éléments dans votre projet.

Vous avez un (ou plusieurs) ***skybox*** dans le jeu, vous allez en assigner un dans les options de la scène.

7. Dans la barre de menus, allez dans :

Window | Lighting

Lighting (Scene)

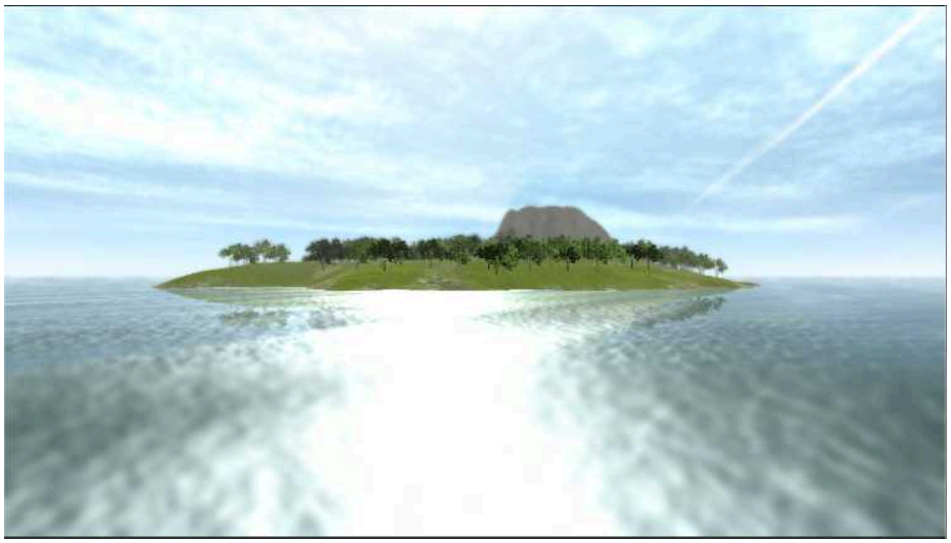
Notez-bien!

*Auparavant, un menu dédié à l'ambiance (nommé : **Render settings**) de la scène était disponible. Depuis la version 5, ce menu a été retiré et remplacé par un menu plus complet et moderne. Celui-ci utilise la technologie **Enlighten** par **Geomerics** pour l'éclairage et l'ambiance globale de la scène.*

7.1. Dans la catégorie : ***Scene,*** changez les paramètres suivants :

Environment Lighting

7.1.1. ***Skybox*** : Dans le skybox que vous avez téléchargé, on utilise : **sky5X3** par *clandestine studio*.
7.1.2. ***Sun*** : Glissez la lumière **Directional Light** qui se trouve dans votre *Hierarchy*.
7.1.3. ***Ambient Source*** : Skybox
7.1.4. ***Ambient Intensity*** : 1
7.1.5. ***Ambient GI*** : Realtime
7.1.6. ***Reflection Source*** : Skybox
7.1.7. ***Resolution*** : 128
7.1.8. ***Reflection Intensity*** : 0.3
7.1.9. ***Reflection Bounces*** : 2



Vous avez maintenant un ciel dans la scène. Ajoutez du brouillard

7.2. Dans le même panneau, changez les paramètres suivants pour obtenir un léger brouillard qui se fond à l'horizon.

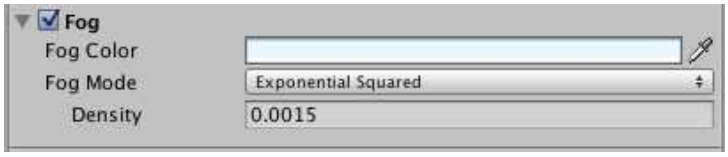
Fog

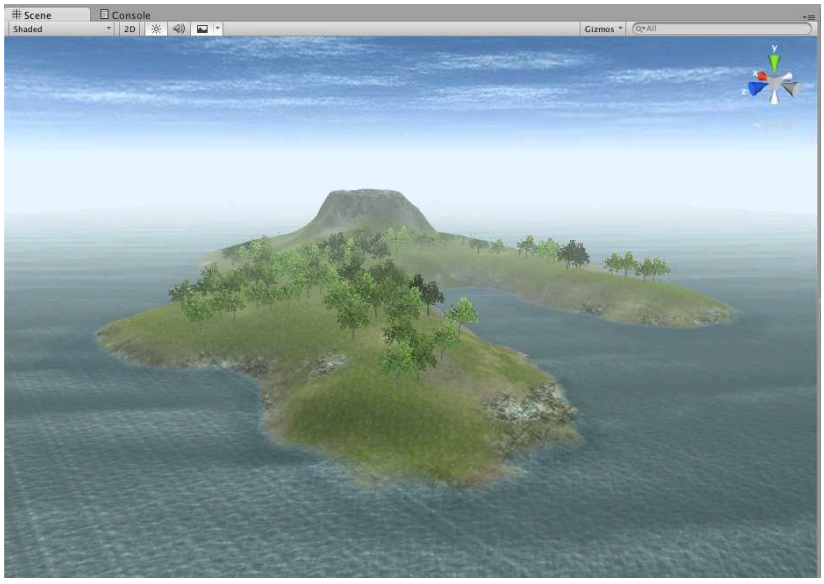
7.2.1. ***Fog*** **:** oui

7.2.2. ***Fog Color*** **:** Avec la ***pipette***, sélectionnez la couleur de l'horizon du ***skybox.***

7.2.3. ***Fog Mode*** **:** Exponential Squared

7.2.4. ***Density*****:** 0.0015

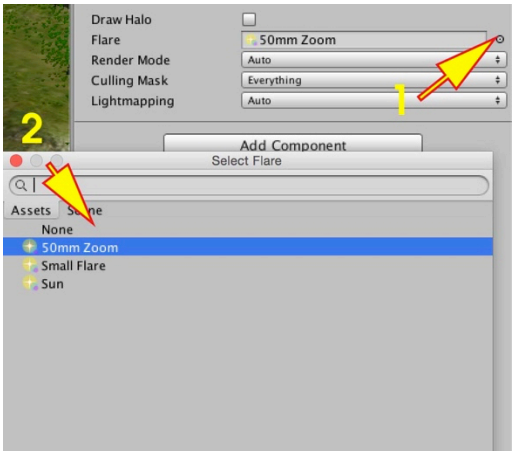




Le ciel devrait se fondre doucement à l'horizon.

Avant de conclure cet exercice, vous allez ajouter un soleil dans le ciel.

8. Sélectionnez l'objet : **Directional Light** dans *Hierarchy.*

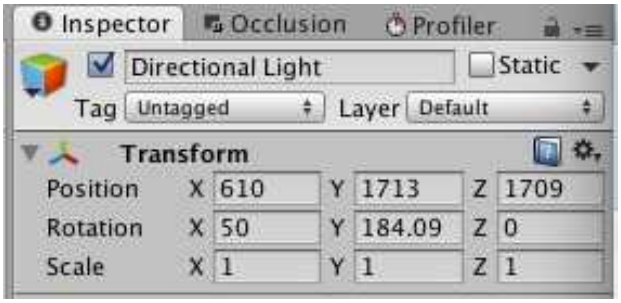


8.1. Changez le paramètre:

Light
8.1.1. ***Flare*** **:** 50mm Zoom

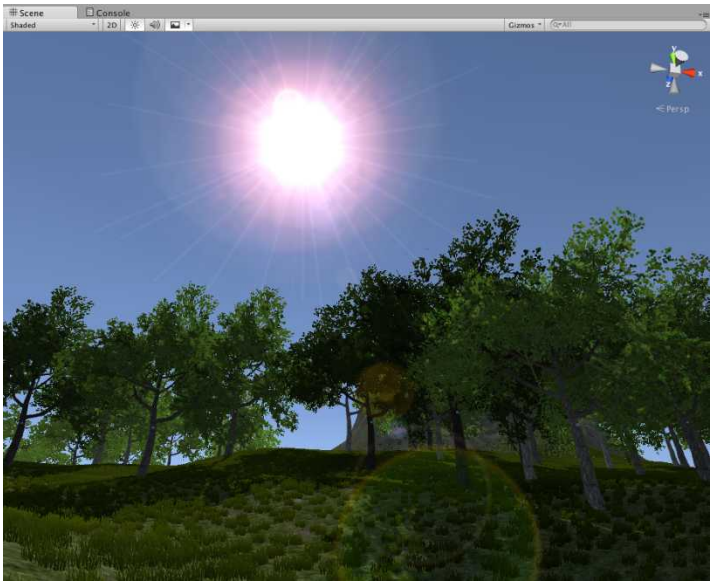
9. Vous avez un point de repère dans le ciel, orientez le point de vue dans la scène afin de voir le soleil sur votre ***Skybox***. Assurez-vous que celui-ci est placé entre le terrain et le ciel, et non en-dessous du sol, sinon le ***Flare*** ne sera pas visible.

10. Ajustez la ***Rotation*** **X** et **Y** de la lumière afin d'aligner le soleil du ***Skybox*** avec le ***Flare*** de la lumière. Avec le ciel : **Sky5X3** les valeurs suivantes semblent correctes.



Transform
10.0.1. ***Position*** **: X=** 610**, Y=** 1713**, Z=** 1709
10.0.2. ***Rotation*** **: X=** 50.93**, Y=** 184.09**, Z=** 0

Tentez d'obtenir un résultat comme ceci :



À la dernière étape, vous allez mettre un avatar dans la scène afin de pouvoir vous promener.

11. Dans *Project,* recherchez : **RigidBodyFPSController**. On le retrouve dans le répertoire :

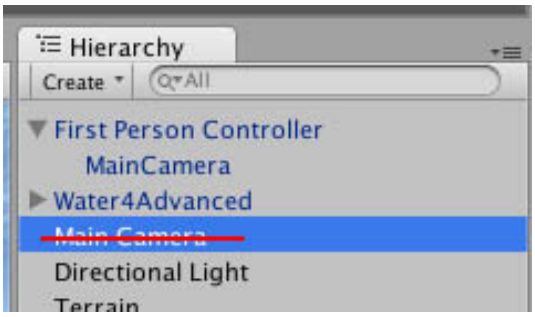
Standard Assets | Characters | FirstPersonCharacter | Prefabs

12. Glissez : **RigidBodyFPSController** dans votre scène.

12.1.Renommez-le : **First Person Controller.**

12.2. Ajustez manuellement la ***Position*** de l'avatar.

13. Supprimez **Main Camera** à la racine de la scène, car vous en avez déjà une dans votre **First Person Controller**.



14. Sélectionnez **Main Camera** qui se trouve dans **First Person Controller**.

14.1. Appuyez sur le bouton « Add Component », puis :

Rendering | Flare Layer

Ce petit changement permet de voir l'éblouissement du soleil avec cette caméra.

Sauvez votre scène (**CTRL+S**) ou (⌘**+S**) sur Mac.

Testez votre jeu.

Retournez en mode édition.

Vous avez une scène bien remplie, mais il est possible de la rendre encore plus réaliste avec des effets de lentilles comme ceux utilisés au cinéma. Dans *Unity*® ceux-ci sont nommés : ***Image Effects.***

Image Effects

Les ***Images Effects*** ne sont pas une nouveauté d'Unity. C'est uniquement depuis la version 5 qu'on peut les utiliser gratuitement. Ces effets permettent d'ajouter du réalisme dans vos projets. Vous allez en voir quelques-uns.

Rayons de soleil (*Sun shafts*)

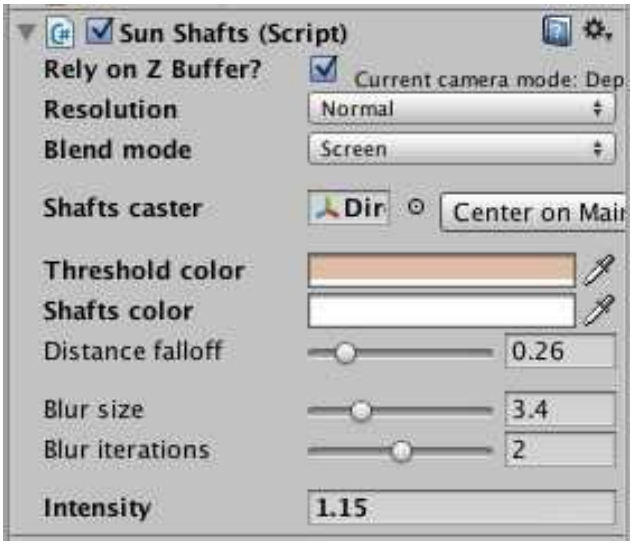
Il faut débuter par les rayons de lumière qui passent au travers la géométrie. Cet effet se nomme : ***Sun Shafts.***

1. Sélectionnez **Main Camera** qui se trouve dans **First Person Controller**.

1.1. Appuyez sur le bouton « Add Component ».

Image Effects | Rendering | Sun Shafts

1.2. Dans l'Inspector, changez les paramètres suivants :



Sun Shafts (Script)

1.2.1. ***Rely on Z Buffer?*** **:** oui
1.2.2. ***Resolution*** **:** Normal
1.2.3. ***Blend mode*** **:** Screen
1.2.4. ***Shafts caster*** **:** Glissez **Directional Light** de *Hierarchy* dans ce champ.
1.2.5. ***Threshold color*** : Beige
1.2.6. ***Shafts color*** **:** Blanc
1.2.7. ***Distance falloff*** : 0.26
1.2.8. ***Blur size*** : 3.4
1.2.9. ***Blur Iterations*** : 2
1.2.10. ***Intensity*** : 1.15

↘ *Sans :* ***rayons de lumière***

Voici le résultat que cet effet ajoutera dans votre scène :

↘ *Avec :* ***rayons de lumière***

Comme vous pouvez le constater, il y a des rayons de lumière qui proviennent du **Directional Light.** N'importe quel ***GameObject*** peut faire l'affaire. Si vous ne l'utilisez pas, les rayons proviendront de tous les angles, là où la couleur tend vers le blanc.

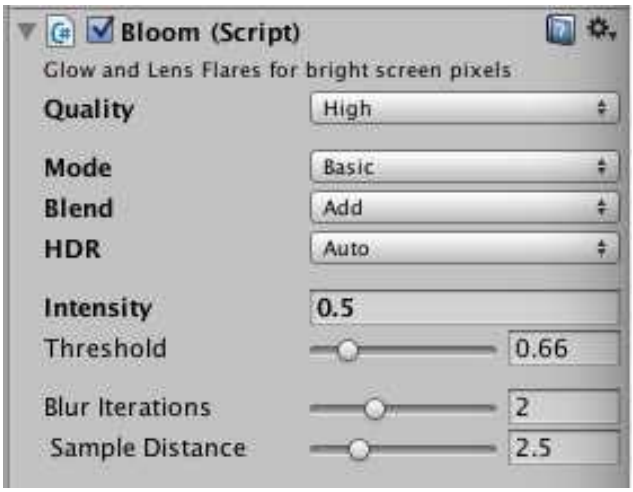
Éblouissement (*Bloom*)

Bloom permet de faire déborder la lumière de sa source pour créer un effet de surexposition.

2. Sélectionnez **Main Camera** qui se trouve dans **First Person Controller**.

2.1. Appuyez sur le bouton : « Add Component ».

Image Effects | Rendering | Bloom and Glow



2.2. Changez les paramètres suivants :

Bloom (Script)
2.2.1. ***Quality*** **:** High
2.2.2. ***Mode*** **:** Basic
2.2.3. ***Blend*** **:** Add
2.2.4. ***HDR*** **:** Auto
2.2.5. ***Intensity*** **:** 0.5
2.2.6. ***Threshold*** **:** 0.66
2.2.7. ***Blur Iterations*** **:** 2
2.2.8. ***Sample Distance*** **:** 2.5

↘ *Sans :* ***éblouissement***

Voici ce que cet effet ajoutera dans votre scène :

↘ *Avec :* ***éblouissement***

Dans cet exemple, on voit l'effet autour des feuilles des arbres qui se mêlent derrière l'éclat de lumière.

L'occlusion ambiante (*Screen Space Ambient Occlusion*)

L'occlusion ambiante représente le dégradé de l'ombrage dans les cavités et autour des surfaces pour simuler un look plus naturel de l'ombrage.

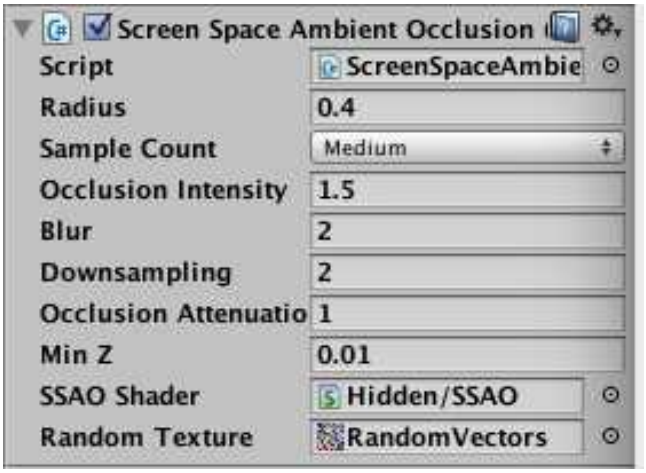
3. Toujours avec **Main Camera** de sélectionné.

3.1. Appuyez sur le bouton « Add Component ».

Image Effects | Rendering | Screen Space Ambient Occlusion

3.2. Changez les paramètres suivants :

Screen Space Ambient Occlusion (Script)
3.2.1. ***Radius*** **:** 0.4
3.2.2. ***Sample Count*** **:** Medium
3.2.3. ***Occlusion Intensity*** **:** 1.5
3.2.4. ***Blur*** **:** 2
3.2.5. ***Downsampling*** **:** 2
3.2.6. ***Occlusion Attenuation*** **:** 1
3.2.7. ***Min*** **: Z=** 0.01



↘ *Sans :* ***occlusion ambiante***

Voici ce que cet effet ajoutera dans votre scène :

↘ *Avec :* ***occlusion ambiante***

Observez l'effet autour des brins de gazon et dans l'ombrage des arbres.

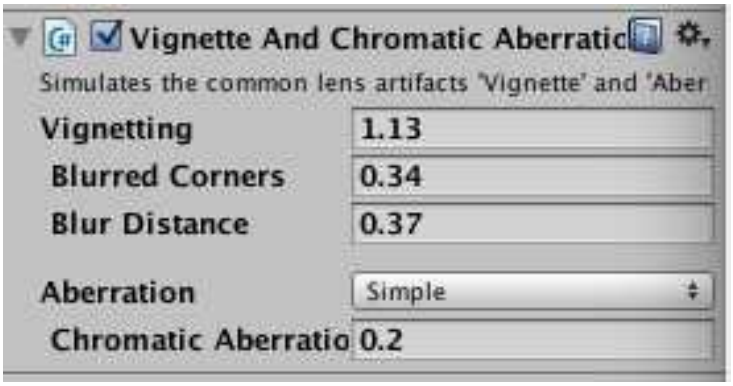
Vignettage et aberration chromatique (*Vignette And Chromatic Aberration*)

Le vignettage est l'assombrissement des coins d'une image. L'aberration chromatique est un effet optique qui produit différentes mises au point, en fonction de la longueur d'onde. En d'autres mots, les couleurs sont déphasées les unes par rapport aux autres.

4. Toujours avec **Main Camera** de sélectionné.

4.1. Appuyez sur le bouton « Add Component ».

Image Effects | Camera | Vignette And Chromatic Aberration



4.2. Changez les paramètres suivants :

Vignette And Chromatic Aberration (Script)
4.2.1. ***Vignetting*** **:** 1.13
4.2.2. ***Blurred Corners*** **:** 0.34
4.2.3. ***Blur Distance*** **:** 0.37
4.2.4. ***Aberration*** **:** Simple
4.2.5. ***Chromatic Aberration*** : 0.2

↘ *Sans :* ***vignettage***

Voici ce que cet effet ajoutera dans votre scène :

↘ *Avec :* ***vignettage***

Vous remarquerez dans l'exemple précédent que les coins sont maintenant assombris.

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Testez votre jeu.

C'est terminé pour cette partie de l'exercice.

Retournez en mode édition.

Conseil!

Maintenant, vous pouvez ajouter d'autres éléments pour améliorer le visuel et l'habillage sonore afin de bonifier l'ambiance de la scène.

Chapitre 8 - Le terrain : partie 2

Matière couverte dans ce chapitre

- **Création de l'intérieur du volcan**
- **Utilisation de systèmes de particules**
- **Création d'un radar**
- **Utilisation de multiples caméras dans une scène.**
- **Utilisation d'occlusion**

Nous allons élaborer le visuel pour le terrain du dernier chapitre. Les nouveautés rencontrées incluront l'utilisation des particules animées ainsi qu'un radar utilisant l'occlusion qui affichera une seule portion de la carte topographique à la fois.

Avant de débuter

Vous avez besoin du paquet : **Chapitre8-9.unitypackage** qui contient tous les éléments graphiques de l'exercice.

Atelier pratique

Raffinement du terrain

Ouvrez le projet **Terrain** du chapitre précédent, puis :

Ouvrez la scène : **Game**

1. Importez les fichiers de l'exercice dans le projet :

1.1. Dans un espace vide du panneau *Project*, faites un clic secondaire, puis :

- **Import Package > Custom Package…**

1.2. Sélectionnez **Chapitre8-9.unitypackage** et importez son contenu.

Vous devez voir ces répertoires ainsi que leur contenu :

FBX
- **radar_occluder.fbx**

Fonts
- **CharisSILMali-B.ttf**

Materials
- **No Name**
- **radar_occluder-No Name**

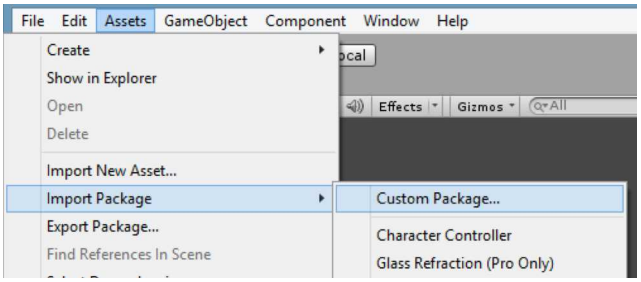
SFX
- **Crac.wav**
- **DiveUnderwater.wav**
- **scream_in_pain.mp3**
- **Pause.wav**

Shaders
- **Alpha-Cancel.Shader**

Textures
- **flint.tga**
- **flint_norm.tga**
- **lavagradient.psd**
- **lava.jpg**
- **lava - normal.tga**
- **Lava particles.png**
- **SmokeSheet.png**
- **menu_btn.png**
- **quit_btn.png**
- **transparent_texture.tga**
- **Maps_arrow.tga**

Notez-bien!

*Vous pouvez également importer un **package** à partir de la barre de menus :*



Assets | Import Package | Custom Package...

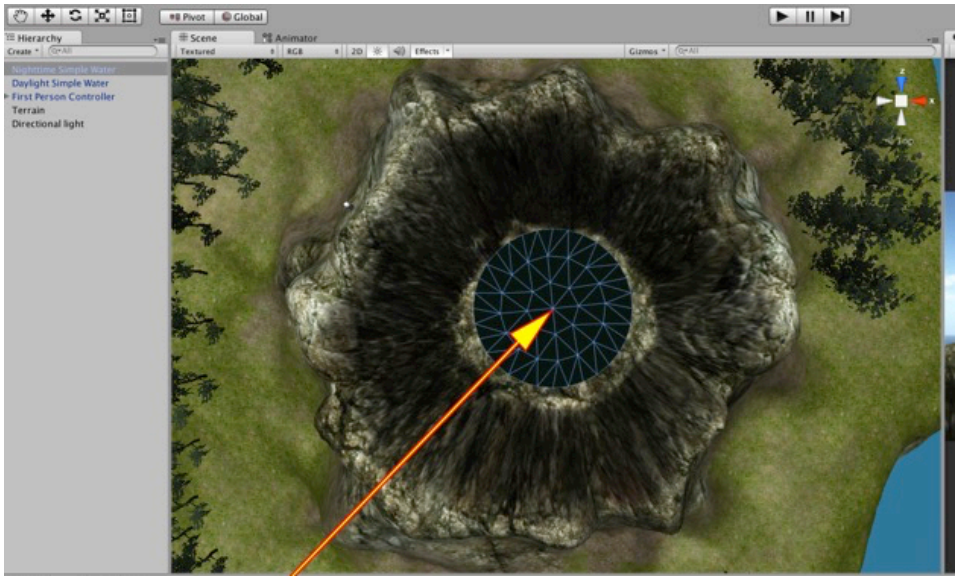
Intérieur du volcan : création de la lave

2. Déplacez votre point de vue dans la *Scene* afin d'être au-dessus du volcan.

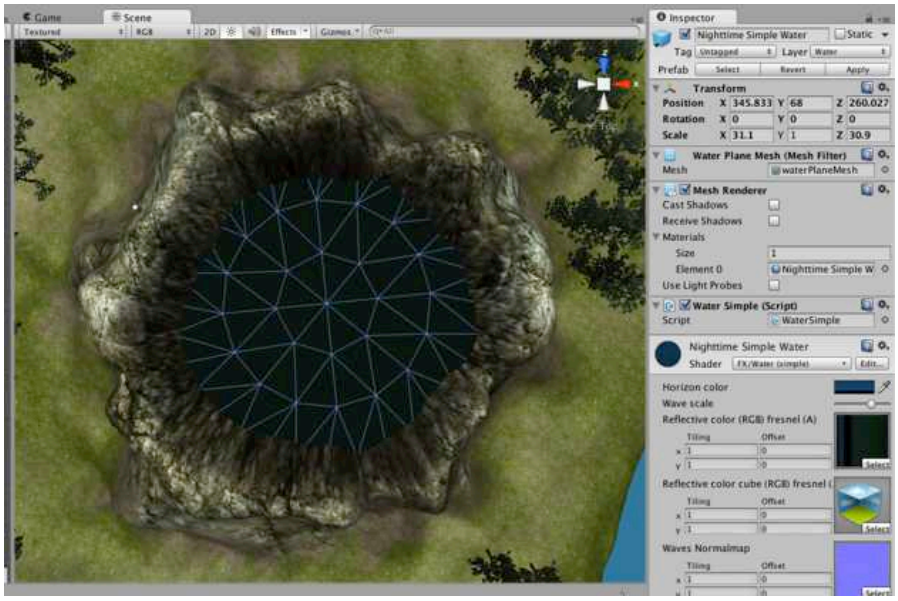
2.1. Si ce n'est déjà fait, cliquez sur les lignes en-dessous du **Gizmo** pour vous mettre en vision isométrique.

3.Dans le panneau *Project,* recherchez le ***prefab*** nommé : **WaterBasicNightime.**

4.Glissez cet objet à l'intérieur du volcan.



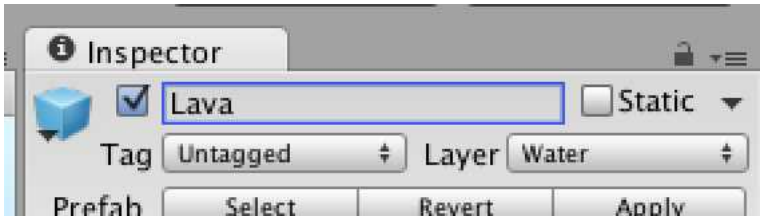
4.1. Repositionnez et redimensionnez cet objet dans la scène, comme ceci :



Transform

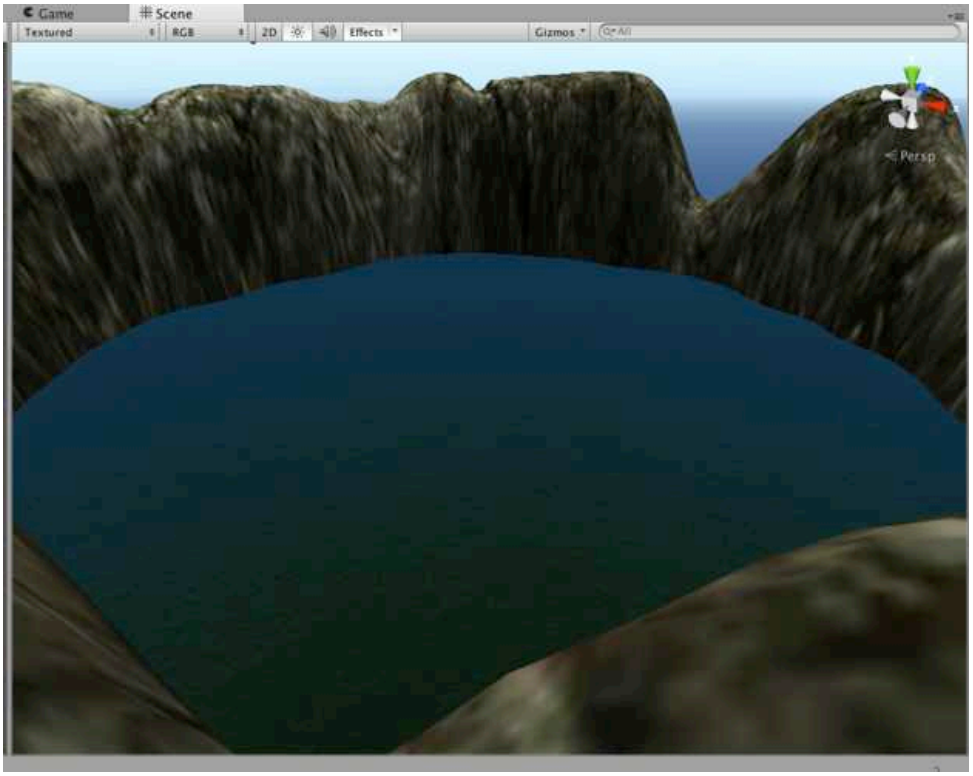
4.1.1. ***Position* : Y**= 69.6

4.1.2. ***Scale* : X et Z** ajustez la taille afin que l'eau remplisse le cratère.



4.2. Renommez cet étendue d'eau : **Lava.**

Assurez-vous d'obtenir un résultat similaire à ceci :



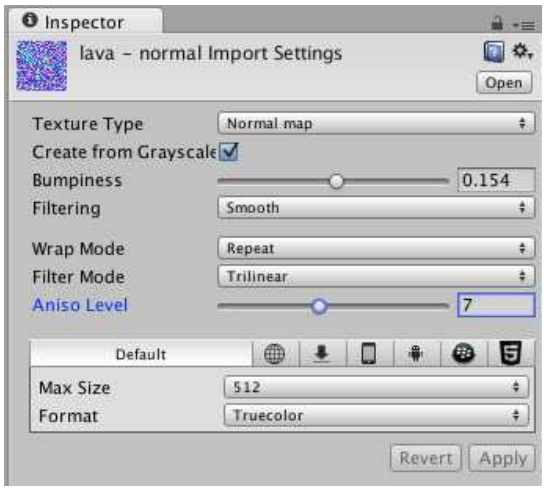
Vous allez changer la couleur de l'eau pour qu'elle devienne orange comme la couleur de la lave.

5. Sélectionnez la texture **lava - normal** dans le répertoire : **Textures** du panneau : *Project.*

5.1. Dans *l'Inspector,* changez :

5.1.1. ***Texture Type*** : Normal map
5.1.2. ***Create from Grayscale*** : non
5.1.3. ***Wrap Mode*** **:** Repeat
5.1.4. ***Filter Mode*** **:** Trilinear
5.1.5. ***Aniso Level*** **:** 7
5.1.6. ***Max Size*** **:** 512
5.1.7. ***Format*** **:** Truecolor

5.2. Appuyez sur « Apply ».

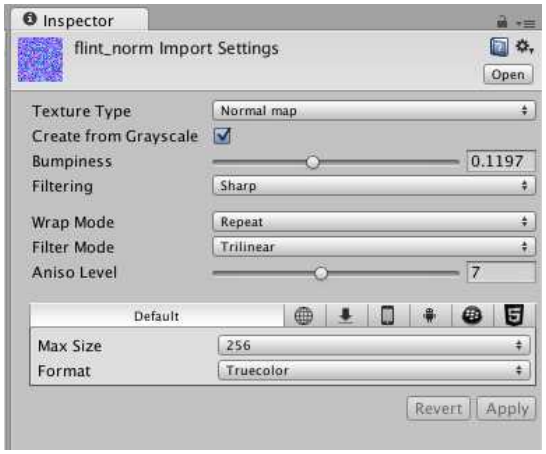


6. Sélectionnez la texture **Flint_norm** dans *Project.*

6.1. Dans *l'Inspector,* changez :

6.1.1. ***Texture Type*** : Normal map
6.1.2. ***Create from Grayscale*** : oui
6.1.3. ***Bumpiness*** **:** 0.12
6.1.4. ***Filtering*** **:** Sharp
6.1.5. ***Wrap Mode*** **:** Repeat
6.1.6. ***Filter Mode*** **:** Trilinear
6.1.7. ***Aniso Level*** **:** 7
6.1.8. ***Max Size*** **:** 256
6.1.9. ***Format*** **:** Truecolor

6.2. Appuyez sur « Apply ».

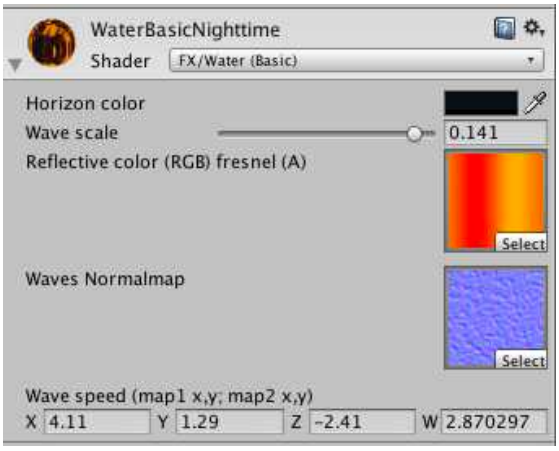


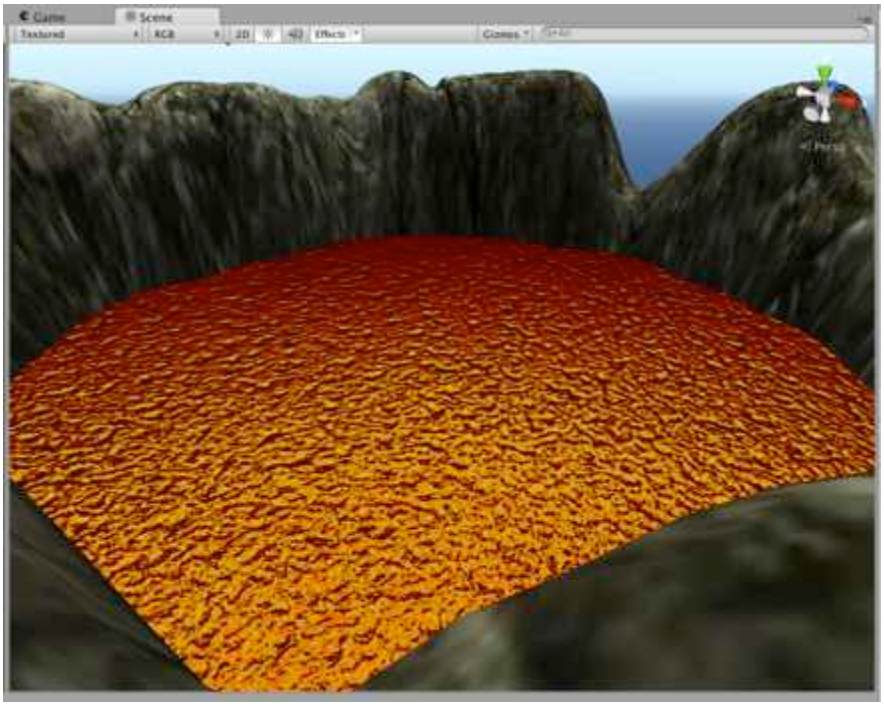
À présent, vous allez modifier les textures de l'eau pour qu'elles deviennent de la lave.

7. Sélectionnez : **Lava** dans votre *Hierarchy.*

7.1. Dans les propriétés ***Material,*** changez :

7.1.1. ***Horizon color*** : noir
7.1.2. ***Wave scale*** : 0.141
7.1.3. ***Reflective color (RGB) fresnel (A)*** : lavagradient
7.1.4. ***Waves Normalmap*** : lava_norm
7.1.5. ***Wave speed*** : **X=** 4.11, **Y=** 1.29, **Z=** -2.41, **W=** 2.87





Vous devriez obtenir ce résultat:

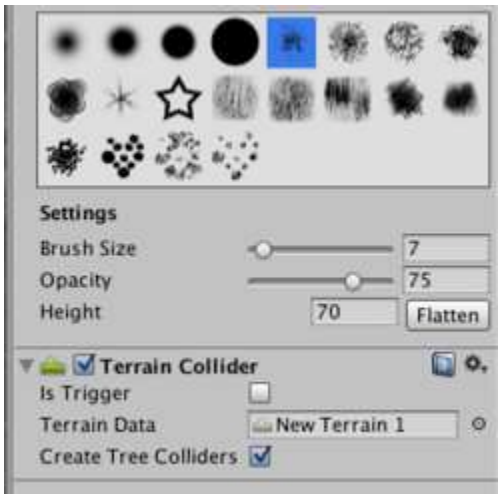
C'est bien, mais une lave avec des plaques de roches en fusion ce serait encore mieux!

Pour ce faire, vous allez modifier la topologie à l'intérieur du volcan afin de faire ressortir le fond à la surface.

8. Sélectionnez le **Terrain** dans *Hierarchy.*

8.1. Dans *l'Inspector*, sélectionnez l'outil : ***Paint height.***

8.2. Dans la section ***Settings***, changez :



8.2.2. ***Brushes*** : Particules
8.2.3. ***Brush Size***: 7
8.2.4. ***Opacity*** : 75
8.2.5. ***Height*** : 70

Notez-bien!

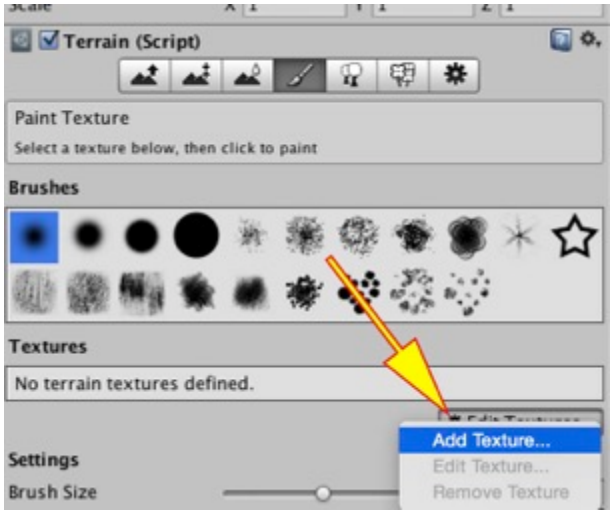
*Le paramètre **height** a été réglé à 70 m car la lave devrait être à une hauteur de 69.6 m. Après avoir remonté le fond du volcan, si vous ne voyez toujours pas les plaques à la surface, ajustez la hauteur de la lave en conséquence.*

9. Maintenant, peignez l'intérieur du volcan afin de faire ressortir des plaques à la surface. Il ne faut pas couvrir entièrement la lave.

Il s'agit maintenant de changer la texture de la roche volcanique.

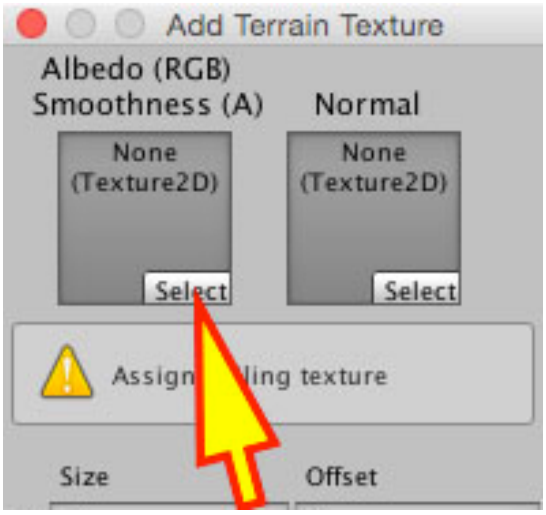
10. Sélectionnez l'outil ***Paint texture***.

11. Dans *l'Inspector*, appuyez sur le bouton « Edit Textures », puis « Add Texture… »



La fenêtre *Add Terrain Texture* va s'ouvrir.

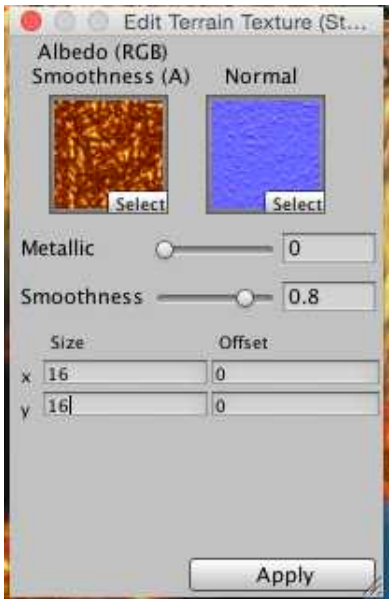
12. Pour obtenir la texture désirée, appuyez sur le bouton « Select » en-dessous du premier espace.



Une seconde fenêtre va s'ouvrir.

13. Recherchez la texture **Lava** et sélectionnez-la.

14. Appuyez sur le bouton « Select » en-dessous du deuxième espace pour obtenir la texture ***Normal Map***.



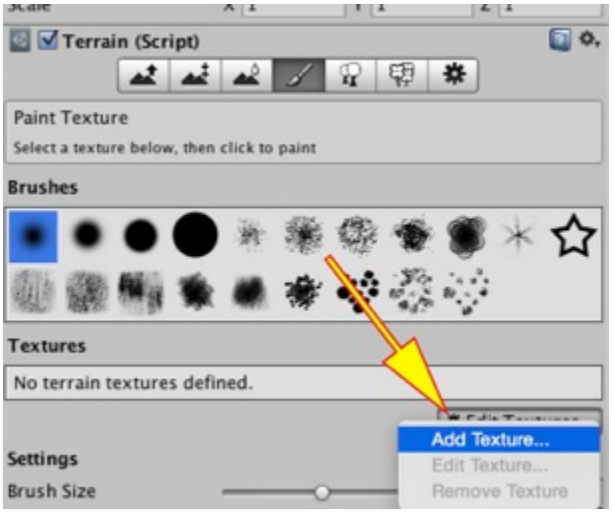
15. Sélectionnez la texture **Lava - normal** que vous avez préparée plus tôt.

15.1. De retour dans la fenêtre *Edit Terrain Texture*, changez :

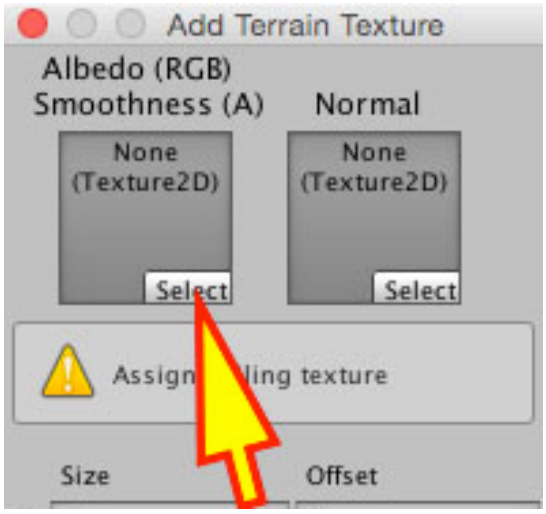
15.1.1. ***Size*** : **X**=16, **Y**=16
15.1.2. ***Metallic*** : 0
15.1.3. ***Smoothness*** : 0.8

15.2. Appuyez sur le bouton « Apply ».

16. Refaites les mêmes étapes dans *l'Inspector* **:**



16.1. Appuyez sur le bouton « Edit Textures », puis « Add Texture… »



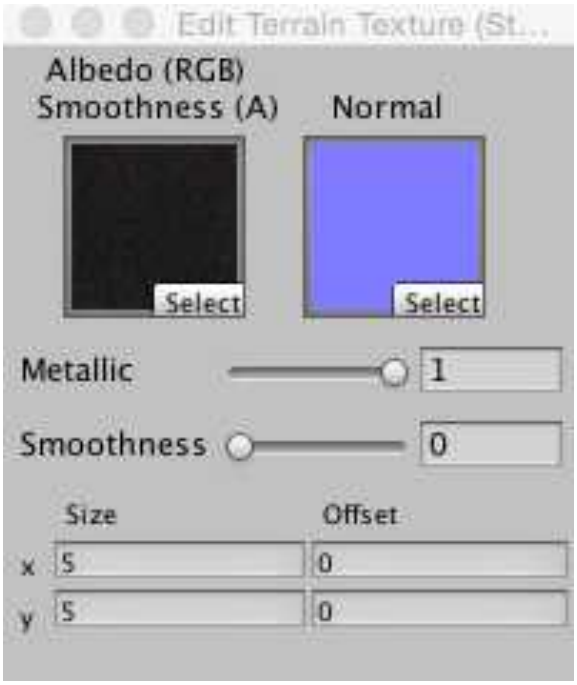
17. Pour obtenir la texture désirée, appuyez sur le bouton « Select » en-dessous du premier espace.

Une seconde fenêtre va s'ouvrir.

18. Recherchez la texture : **flint** et sélectionnez-la.

19. Appuyez sur le bouton « Select » en-dessous du deuxième espace pour obtenir la texture ***Normal Map***.

20. Sélectionnez la texture **flint_norm** que vous avez préparée plus tôt.



20.1. De retour dans *Edit Terrain Texture*, changez :

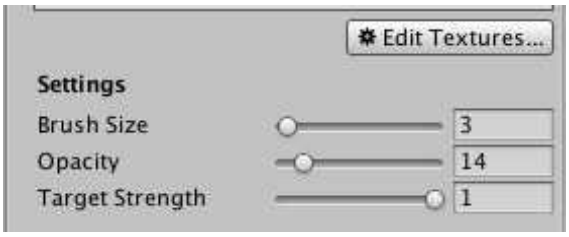
20.1.1. ***Size*** : **X**=5, **Y**=5
20.1.2. ***Metallic*** : 1
20.1.3. ***Smoothness*** : 0

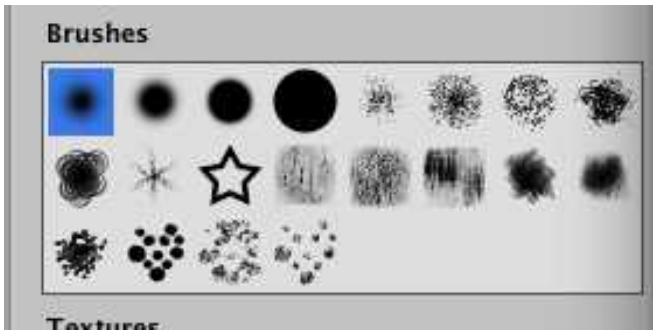
20.2. Appuyez sur « Apply ».

21. Sélectionnez la texture obtenue dans la liste.

21.1. Changez les paramètres du pinceau pour :

21.1.1. ***Brushes*** : Cercle dégradé
21.1.2. ***Brushes Size*** : 3
21.1.3. ***Opacity*** : 14
21.1.4. ***Target Strength*** : 1





22. Peignez les plaques à l'intérieur du volcan comme ceci :

Pour ajouter du réalisme, placez une lumière à l'intérieur du volcan.

23. Dans la barre de menus, allez dans :

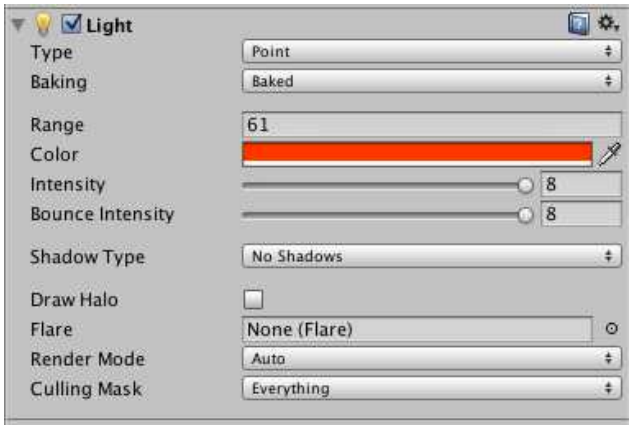
GameObject | Light | Point light

23.1. Nommez-la **LavaLight**.

23.2. Positionnez votre lumière au-dessus de la lave du volcan.

23.3. Changez les paramètres suivants :

Light
23.3.1. ***Baking* :** Baked
23.3.2. ***Range***: 61
23.3.3. ***Color***: Rouge/orangé
23.3.4. ***Intensity*** : 8
23.3.5. ***Bounce Intensity*** : 8
23.3.6. ***Shadow Type*** : No Shadows
23.3.7. ***Draw Halo*** : non
23.3.8. ***Flare*** : Aucun
23.3.9. ***Render Mode*** : Auto
23.3.10. ***Culling Mask*** : Everything



Vous avez maintenant la base de votre volcan.

Afin de rendre le tout encore plus réaliste, vous allez ajouter différentes particules afin de créer des effets spéciaux.

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Intérieur du volcan : utilisation des particules

Premièrement, vous utiliserez des particules animées pour créer un effet de bouillonnement.

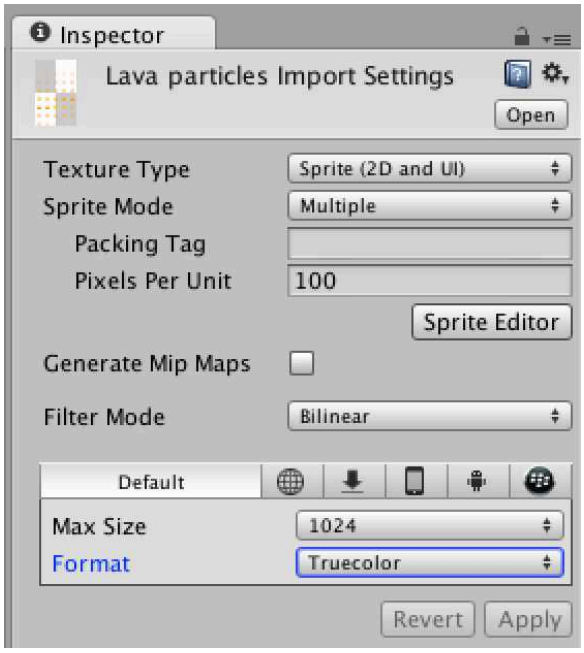
Vous utilisez la texture **Lava particles.png.**

Comme vous pouvez le constater, cette texture contient une séquence d'images 2D sous forme de grille. On appelle ce type de texture ***Animation sheet*** ou ***Sprite sheet.***

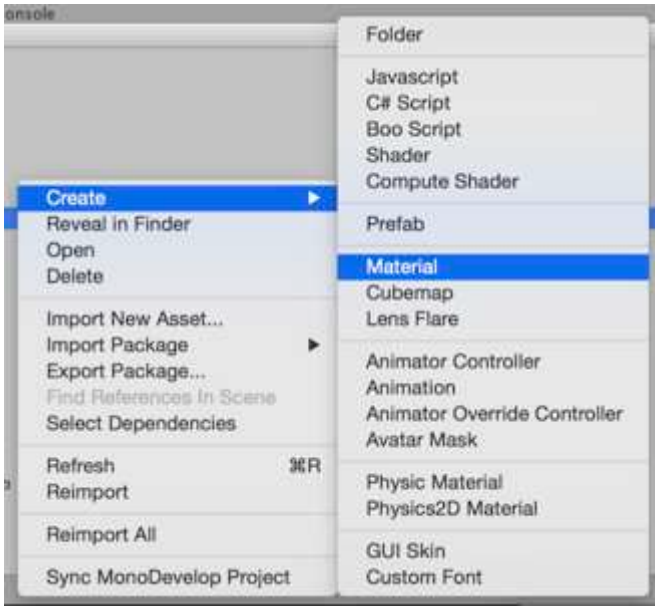
24. Sélectionnez l'image en question dans *Project.*

24.1. Dans *l'Inspector,* changez les propriétés de l'image pour :

24.1.1. ***Texture Type*** : Sprite (2D and UI)
24.1.2. ***Sprite Mode*** : Multiple
24.1.3. ***Format*** : Truecolor



24.2. Appuyez sur le bouton « Apply ».



Il vous faut maintenant sélectionner ***Material*** pour créer vos particules.

25.Placez votre curseur au-dessus de la fenêtre *Project*, puis faites un clic secondaire :

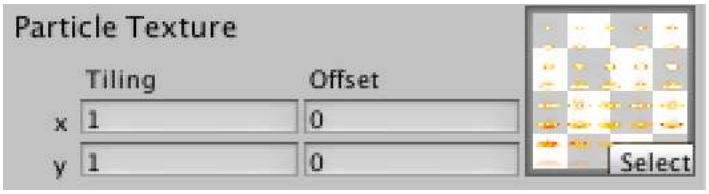
➢ **Create > Material**

25.1. Nommez-le **Lava_particles.**

26. Sélectionnez le ***Material*** et changez la propriété ***Shader*** pour :

Particles | Additive

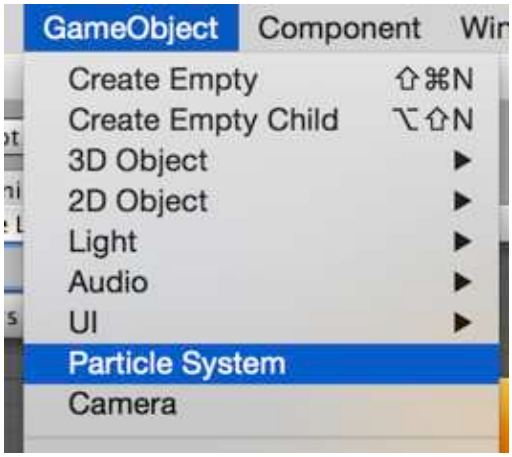
27. Glissez la texture **Lava particles** dans l'espace réservé à cet effet sur ***Material***.



Il est maintenant temps de mettre un émetteur de particules dans la scène.

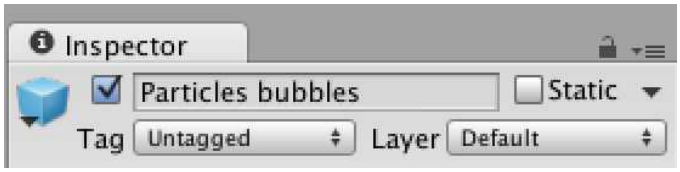
28. Dans la barre de menus, cliquez sur :

GameObject | Particle System



29. Sélectionnez l'émetteur de particules dans *Hierarchy*.

29.1. Nommez-le **Particles bubbles.**

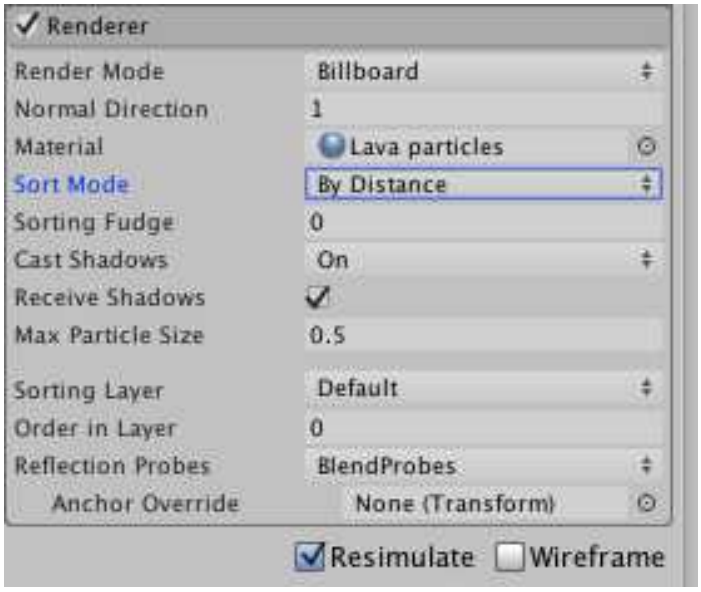


29.2. Dans *Hierarchy*, mettez **Particles bubbles** enfant de l'objet **Lava.**

29.3. Glissez le ***Material*** dans **Lava_particles** sur l'émetteur.

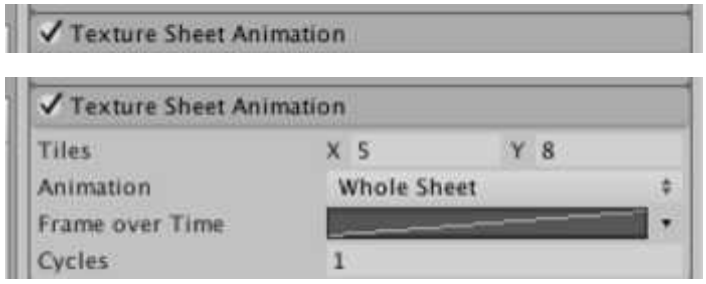
29.4. Changez les paramètres :

Transform
29.4.1. ***Position*** : **X=** 0, **Y=** 0.8, **Z=** 0



Renderer
29.4.2. ***Sort Mode*** : By Distance

Ce paramètre fait en sorte que les particules plus près de la caméra sont placées en avant-plan.



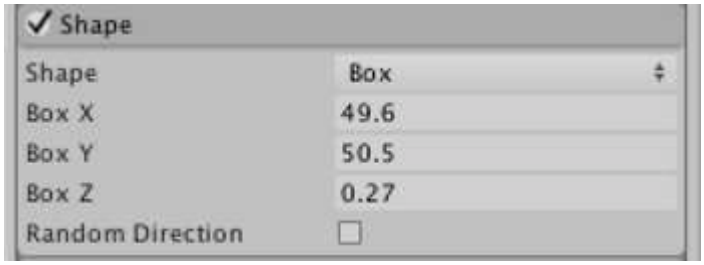
Texture Sheet Animation
29.4.3. Activez cette section.

29.4.4. ***Tiles*** **: X=** 5, **Y=** 8

L'animation des particules devrait être agréable à observer.

En détails...

Remarquez que la valeur **X** représente le nombre d'images **par ligne** et **Y** représente le nombre **par colonne**.



Shape

29.4.5. ***Shape*** : Box

29.4.6. ***Box* : X et Y** : Afin que l'émetteur couvre l'intérieur du volcan.

29.4.7. ***Box* : Z** : 0.27

Voici le résultat que vous devriez obtenir.

Dans les paramètres de bases

29.4.8. ***Duration*** : 5.0

29.4.9. ***Looping*** : Oui

29.4.10. ***Prewarm*** : Oui

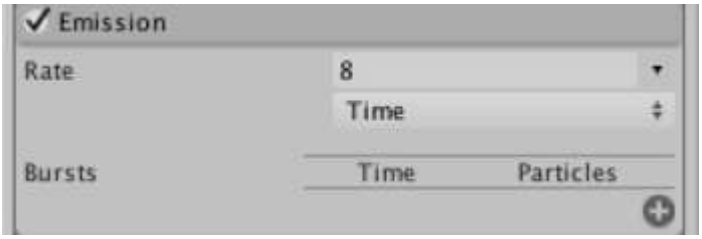
29.4.11. ***Start Lifetime*** : 1

29.4.12. ***Start Speed*** : 0

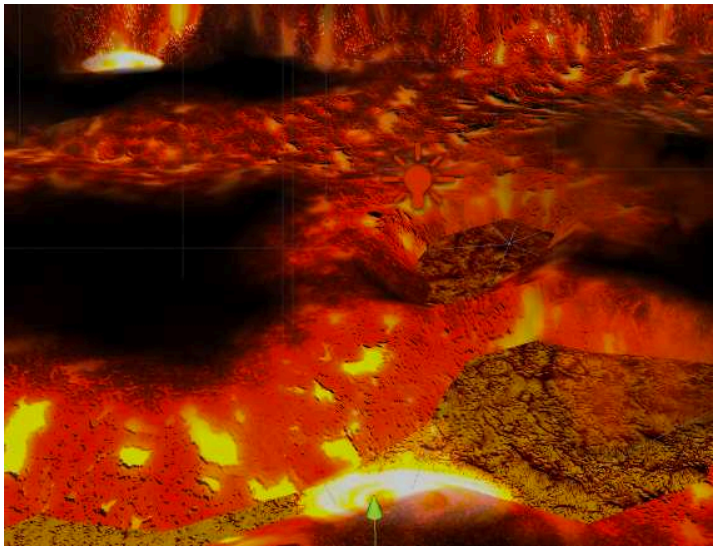
29.4.13. ***Start Size* :** 4.2

Emission

29.4.14. ***Rate*** : 8



Voici le résultat que vous devriez obtenir :



Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Vous allez créer un système de particules de fumée qui émergera de la lave en fusion.

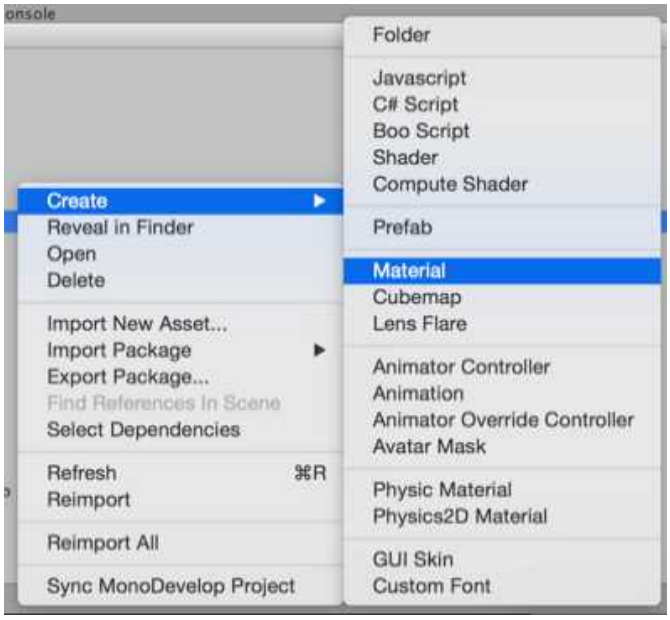
30. Sélectionnez l'image **SmokeSheet** dans *Project.*

30.1. Dans *l'Inspector,* changez les propriétés de l'image pour :

30.1.1. ***Texture Type*** : Sprite (2D and UI)
30.1.2. ***Sprite Mode*** : Multiple
30.1.3. ***Format*** : Truecolor

30.2. Appuyez sur le bouton « Apply ».

Créez maintenant un ***Material*** pour la fumée.



31. Placez votre curseur au-dessus de la fenêtre *Project* et faites un clic secondaire, puis :

➢ **Create > Material**

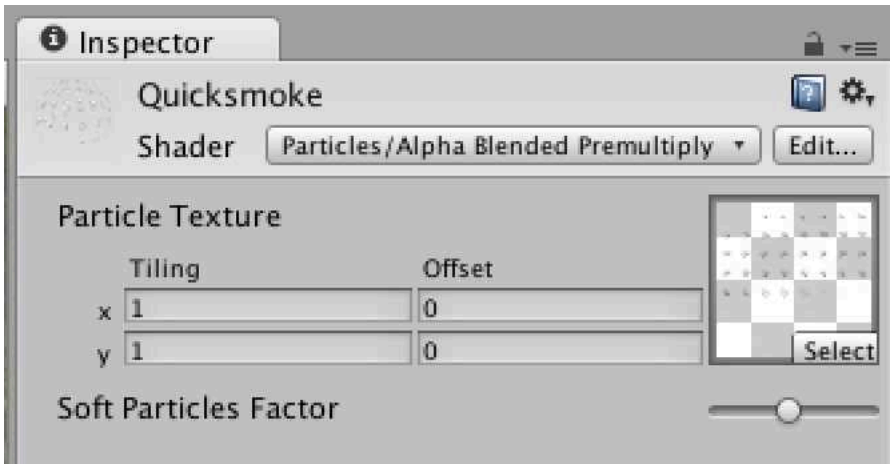
31.1. Nommez-le **Quicksmoke.**

32. Sélectionnez le nouveau ***Material***.

32.1. Dans *l'Inspector*, changez le ***Shader*** pour :

Particles | Alpha Blended Premultiply

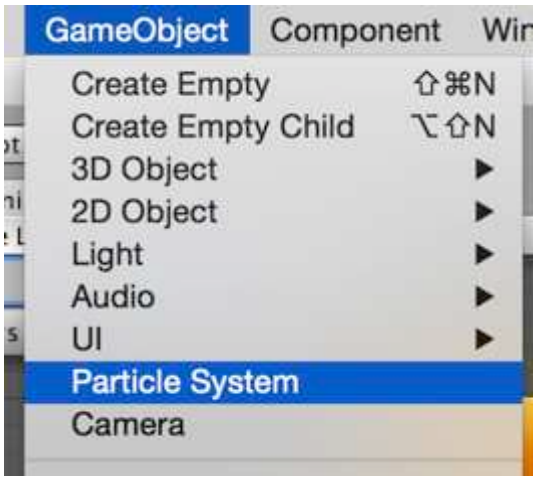
32.2. Glissez la texture **SmokeSheet** dans la case ***Particle Texture*** :



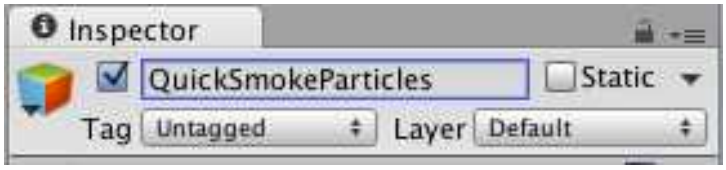
Il faut créer un nouvel émetteur de particules.

33. À partir de la barre de menus, allez dans :

GameObject | Particle System

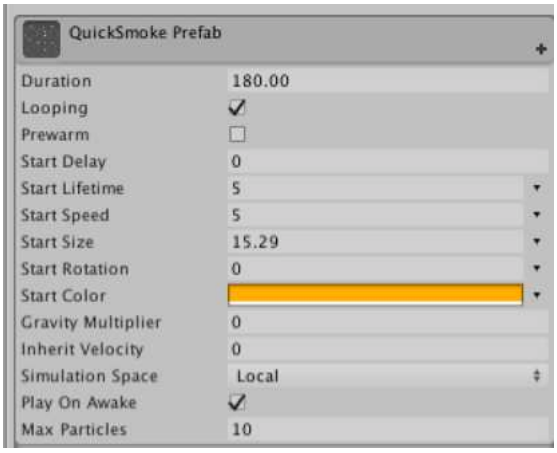


33.1. Nommez cet émetteur **QuickSmokeParticles.**

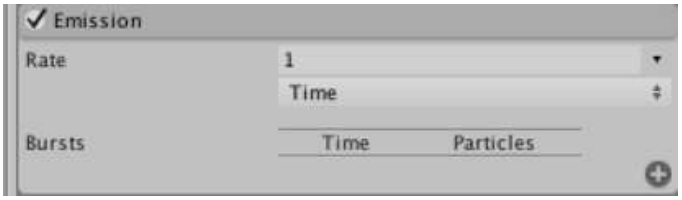


33.2. Glissez le ***Material*** **QuickSmoke** sur l'émetteur **QuickSmokeParticles.**

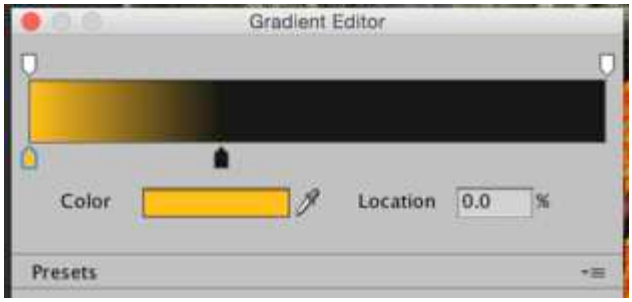
33.3. Dans *l'Inspector*, changez les paramètres suivants :



Paramètres de base:
33.3.1. ***Duration*** : 180
33.3.2. ***Looping*** : Oui
33.3.3. ***Prewarm*** : Non
33.3.4. ***Start Lifetime*** : 5
33.3.5. ***Start Speed*** : 5
33.3.6. ***Start Size*** : 15.29
33.3.7. ***Start Color*** : Orange
33.3.8. ***Max Particles*** : 10



Emission
33.3.9. ***Rate*** : 1

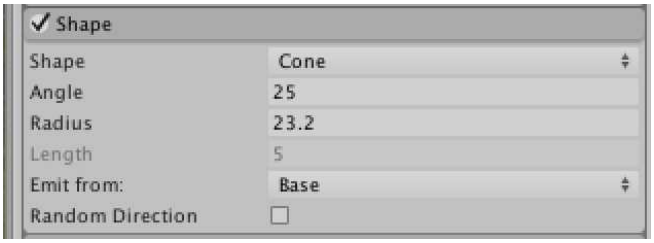


Color over Lifetime
33.3.10. Activez cette section.

33.4. Changez le **dégradé** (Orange au début vers le noir).

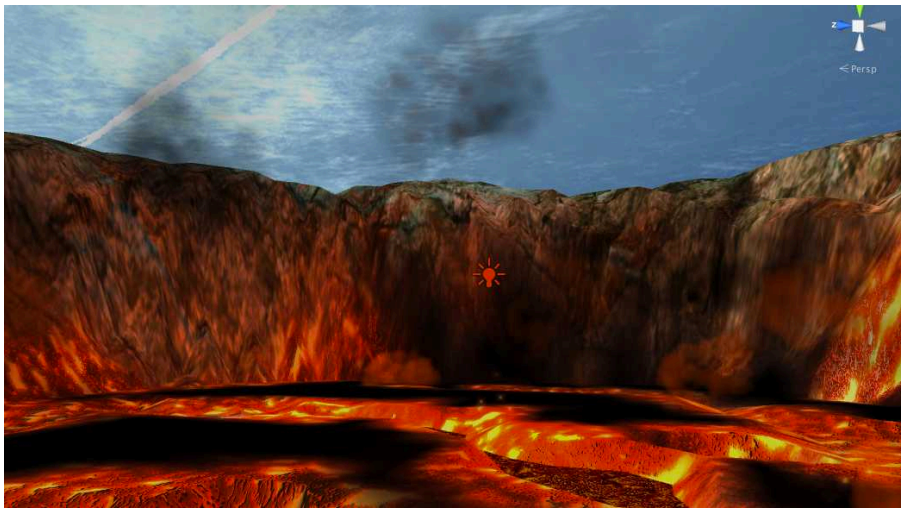
Texture Sheet Animation
33.4.1.Activez cette section.
33.4.2.***Tiles*** : **X**=8, **Y**=8

Shape
33.4.3. ***Shape*** : Cone
33.4.4. ***Angle*** : 25
33.4.5. ***Radius*** : L'émetteur couvre l'intérieur du volcan.

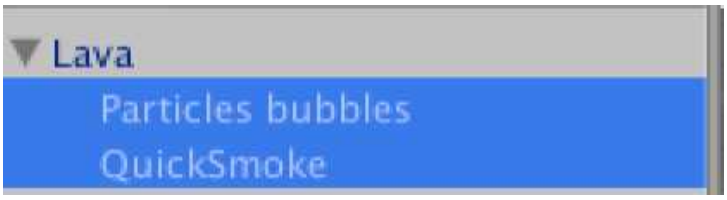


34. Placez l'émetteur juste au-dessus de la lave afin qu'elle ressemble à ceci :

Voici le résultat espéré :



Une fois que l'effet est à votre goût, il faut attacher l'émetteur de fumée à la lave. Par la suite, vous pourrez animer l'objet parent et les particules suivront.

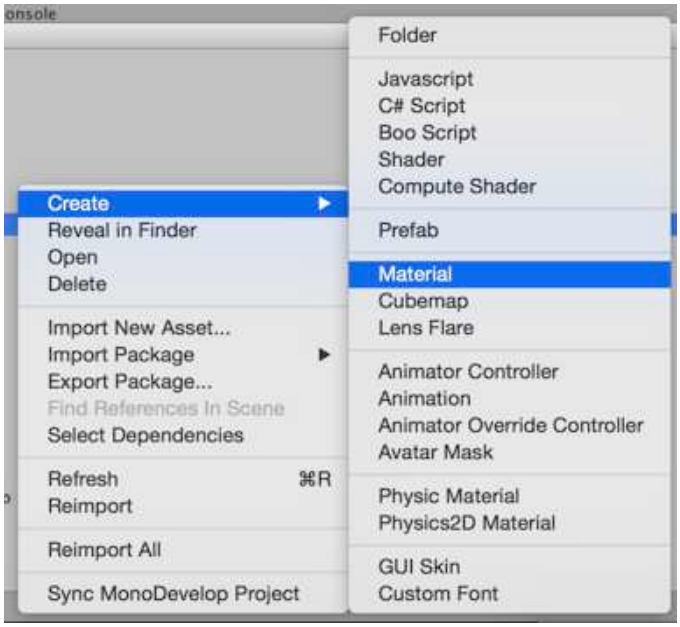


35. Dans *Hierarchy*, mettez l'émetteur de particules **QuickSmoke** enfant de **Lava.**

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Avant d'animer la lave, il faut ajouter un dernier émetteur de particules pour simuler une accumulation de fumée volcanique dans le ciel. Vous utiliserez une texture fournie avec Unity®.

36. Placez votre curseur au-dessus du panneau *Project* et faites un clic secondaire, puis :



➢ **Create > Material**

36.1. Nommez-le **smoke.**

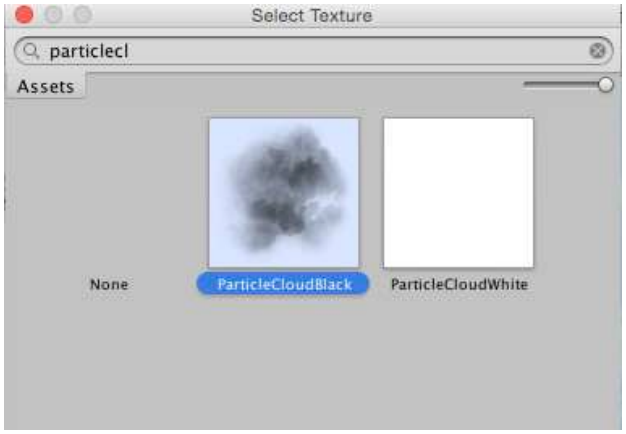
37. Sélectionnez le ***Material*** : **smoke.**

37.1. Dans *l'Inspector*, changez le ***Shader*** :

Mobile | Particles | Multiply

37.2. Pour obtenir le paramètre ***Particle Texture,*** appuyez sur « Select ».

37.3. Trouvez et sélectionnez la texture avec le nom **ParticleCloudBlack.**



Notez-bien!

*Vous vous demandez sûrement pourquoi utiliser un **Shader** (**mobile**) alors qu'il y a un **Shader** équivalent pour PC? La raison est un bug d'affichage entre le **Shader** de feuillage et les **Shaders** transparents pour les particules. C'est ce qui arrive quand on utilise le **Shader** régulier au lieu du **Shader** mobile.*

↘ *Shader : **Particles / Multiply***

↘ *Shader : **Mobile/ Particles / Multiply***

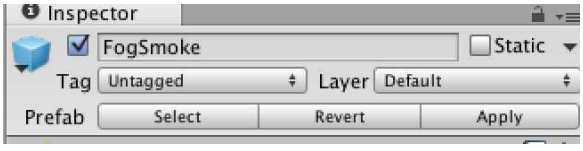
Vous pouvez faire l'expérience vous-même. Il faut espérer que ce problème sera résolu dans une version ultérieure à 5.0.0

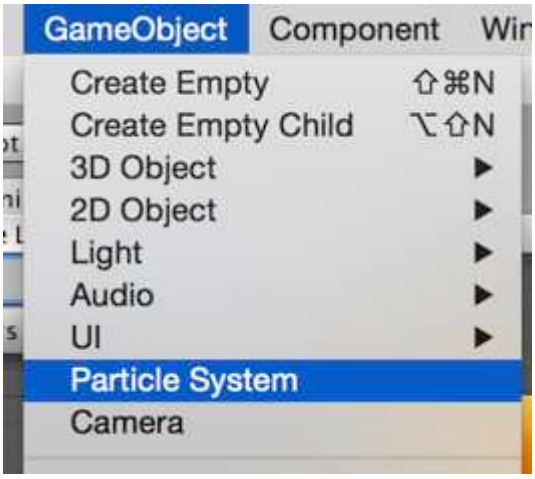
Vous allez ajouter un nouvel émetteur de particules.

38. Dans la barre de menus, allez dans :

GameObject | Particle System

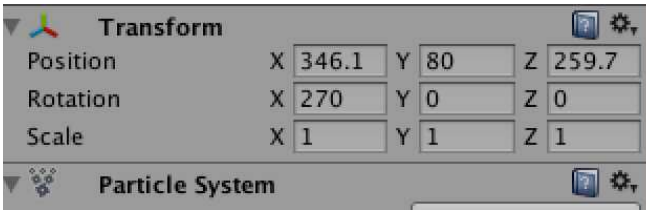
38.1. Nommez-le **FogSmoke.**



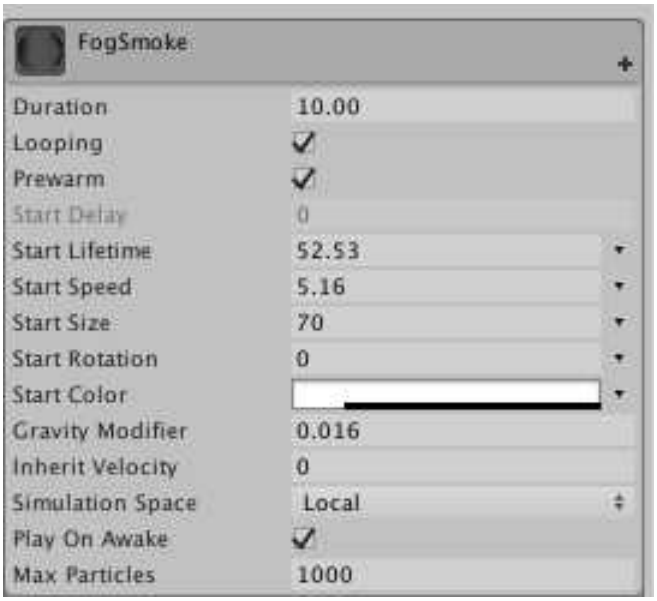


39. Glissez le ***Material* smoke** sur cet émetteur.

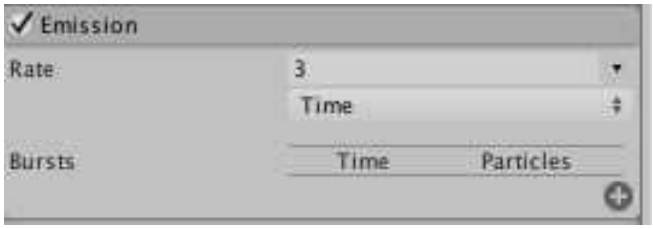
40. Dans *l'Inspector,* changez :



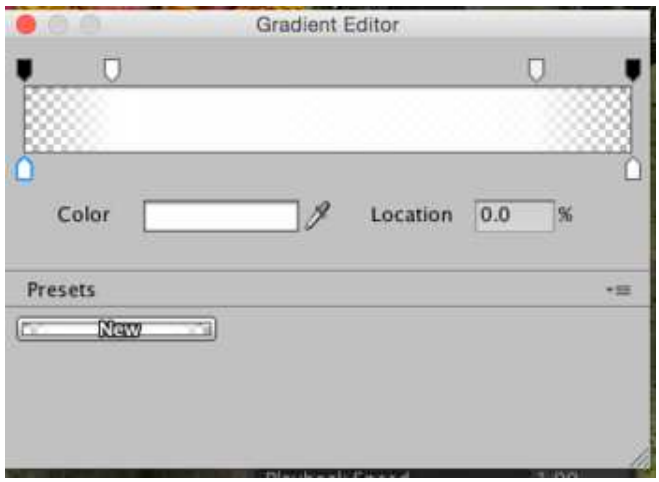
Transform
40.0.1. ***Position* : Y**= 80



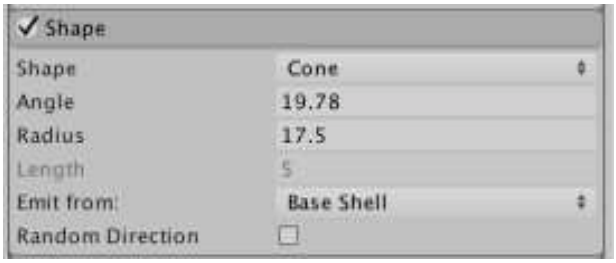
Paramètres de base
40.0.2.***Duration*** : 10.0
40.0.3.***Looping*** : Oui
40.0.4.***Prewarm*** : Oui
40.0.5.***Start Lifetime*** : 50
40.0.6.***Start Speed*** : 5.16
40.0.7.***Start Size*** : 70
40.0.8.***Start Color*** : Blanc, **alpha** = 42
40.0.9.***Gravity Multiplier*** : 0.016
40.0.10.***Simulation Space*** : Local
40.0.11.***Play On Awake*** : oui
40.0.12.***Max Particles*** : 1000



Emission
40.0.13.***Rate***: 3

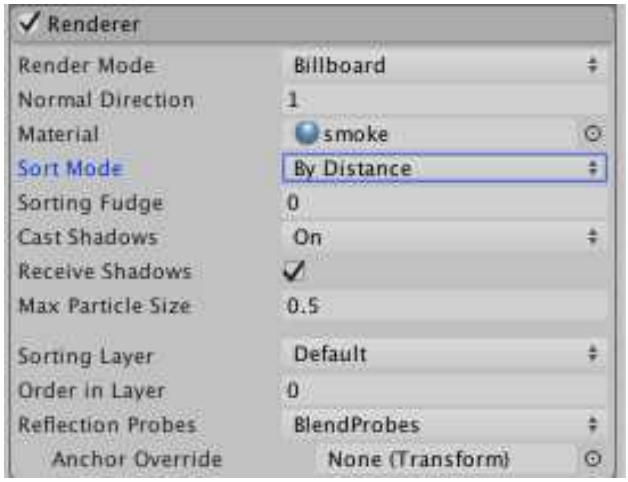


Color over Lifetime
40.0.14.***Color* :** Blanc (Alpha 0%) aux extrémités et (Alpha 100%) dans le milieu.



Shape

40.0.15. ***Shape*** : Cone
40.0.16. ***Angle*** : 19.78
40.0.17. ***Radius*** : L'émetteur couvre l'intérieur du volcan.
40.0.18. ***Emit from*** : Base Shell



Renderer

40.0.19.***Sort Mode*** : By Distance

Tentez d'obtenir ce résultat :

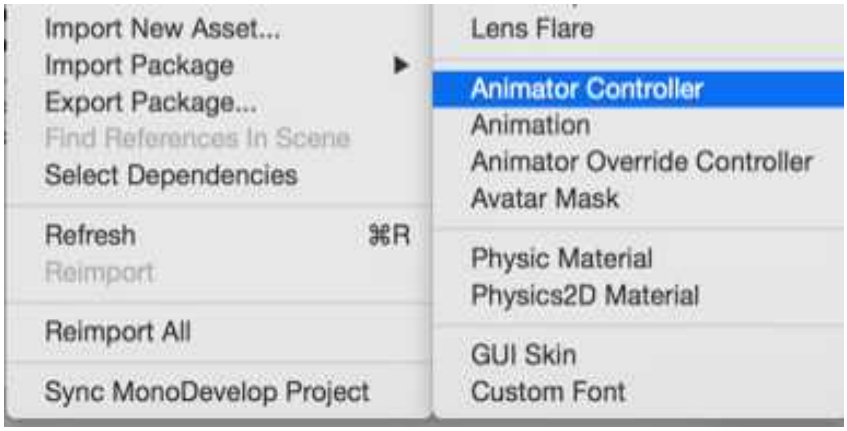
Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Vous avez tous les éléments en place. Maintenant, il faut créer une légère animation de la lave.

Intérieur du volcan : animation de la lave

Depuis la version 4.0 d'Unity®, un nouveau système d'animation du nom de *Mecanim* a fait son entrée. Au tout début, le système avait beaucoup de limitations qui le rendait difficilement utilisable dans une production de jeu. Heureusement, avec les mises à jour rapides du logiciel, Unity® 5 comprend maintenant une version beaucoup plus robuste de l'outil.

Avant de se lancer dans l'animation, vous devez créer une « Animation Controller ». Ce contrôleur sert de canevas pour les différentes animations d'un objet.

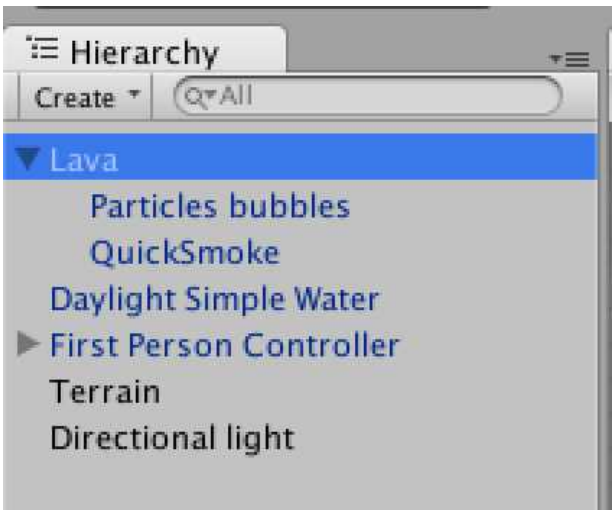


41. Placez votre curseur au-dessus de la fenêtre *Project* et faites un clic secondaire :

➢ **Create > Animator Controller**

41.1. Nommez-le **Volcano.**

Avant d'assigner ce contrôleur à la lave, vous allez ajouter un ***Component*** à l'objet **Lava** qui est le parent des émetteurs : **Particles bubbles** et **QuickSmoke.**



42.Sélectionnez **Lava** dans *Hierarchy.*

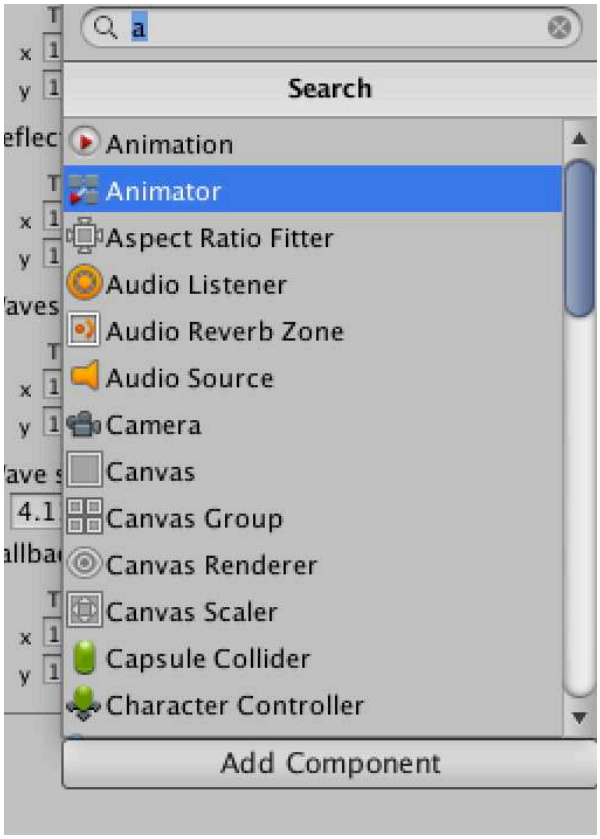
42.1. Dans *l'Inspector,* appuyez sur le bouton « Add Component », puis :

Miscellaneous | Animator

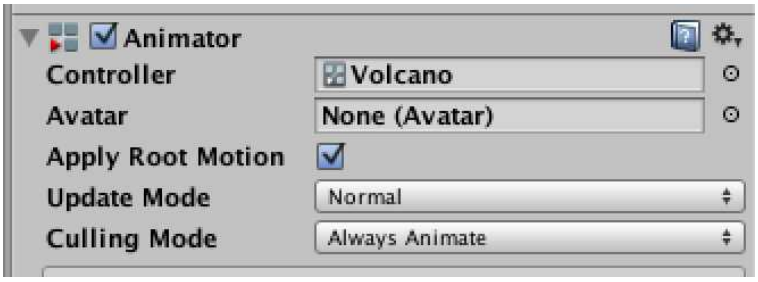
Notez-bien!

*Faites attention, il y a deux components similaires : **Animator** et **Animation**. Le premier sert pour **Mecanim** tandis qu'**Animation** est pour le système **legacy**.*

*Vous utilisez **ANIMATOR** pour cet exercice. Vous ne pouvez pas utiliser les deux.*



43. Glissez le contrôleur **Volcano** dans le champ ***Controller.***



Maintenant que le contrôleur est assigné à la lave, vous allez pouvoir ajouter une animation. Pour cela, il faut utiliser la fenêtre d'animation qui est incluse avec Unity®.

44. Ouvrez la fenêtre d'*animation* avec les touches (**CTRL+6**) ou (**⌘+6**) sur Mac.

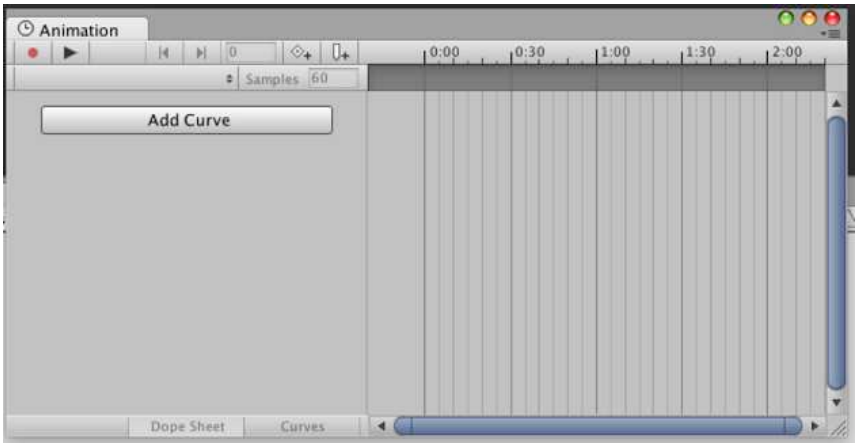
Notez-bien!

Vous pouvez également utiliser la barre de menus :

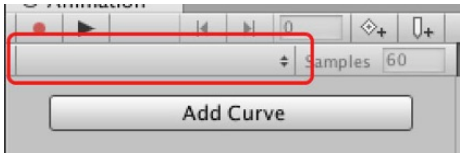
Windows | Animation.

Fenêtre d'animation

La première étape est de créer une nouvelle animation.

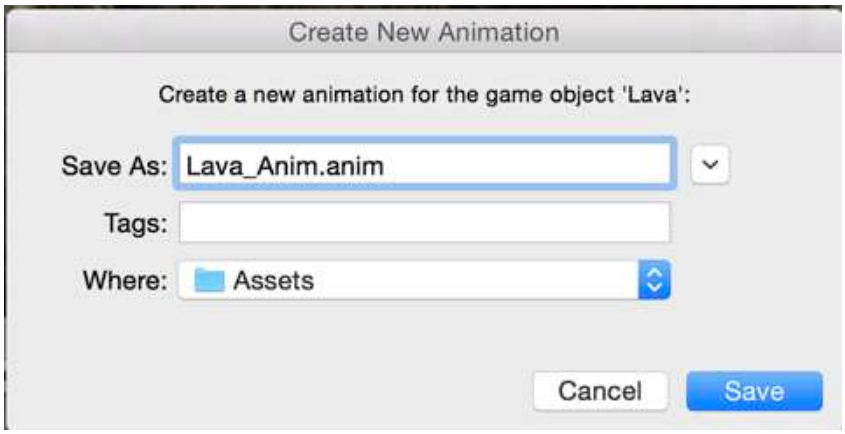


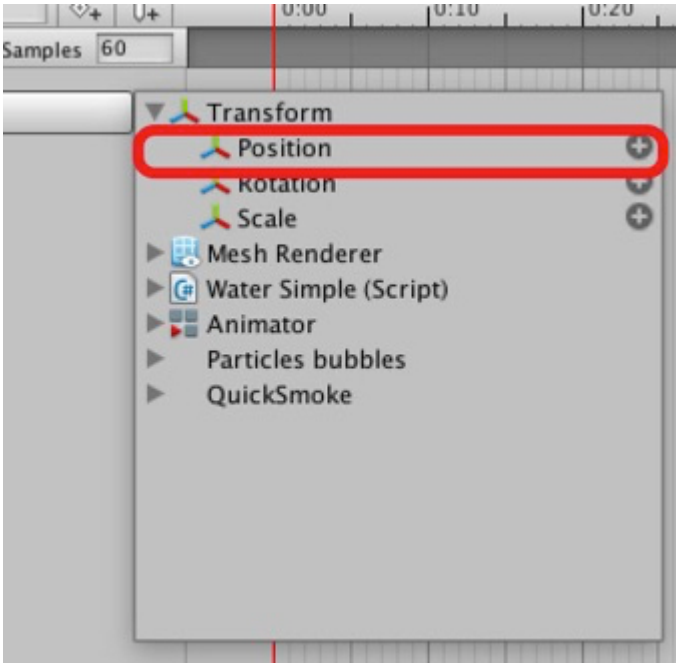
45. Appuyez sur le champ **sans nom** en-dessous des boutons d'enregistrement et de lecture.



46. Puis, appuyez sur « [Create New Clip] ».

47. Sauvez votre clip sous le nom **Lava_Anim.anim** dans le répertoire **Assets**.





Vous devez spécifier à l'outil quelles sont les propriétés à animer.

48. Appuyez sur le bouton « Add Property », puis :

Transform | Position

48.1. Appuyez sur le **(+)** pour créer un champ.

Une fois que l'élément est animé, vous devez définir une clé d'animation.

49. À l'image **0,** appuyez sur le bouton d'ajout de clé « Add keyframe ».

Vous possédez maintenant une image clé. Par défaut, il faut définir une valeur pour l'élément ***Position.y***

49.1. À la droite du texte ***Position.y,*** inscrivez la valeur **69.6** ou une autre valeur appropriée pour votre scène.

49.2. À l'exception de l'image clé, si vous avez d'autres points sur la ligne du temps que vous venez de créer, effacez-les.

Vous avez maintenant une image clé au début de votre ligne du temps, il faut en ajouter une autre beaucoup plus loin.

50. Entrez le nombre **600** dans le champ texte du numéro d'image.

50.1. Appuyez sur le bouton « Add Keyframe ».

En détails...

Vous avez maintenant deux images clé, une au début et une à la fin, avec la même valeur. C'est important que les éléments demeurent ainsi, sinon l'animation ne bouclera pas correctement.

Vous allez ajouter une image clé entre les deux, nous changerons alors sa valeur pour créer l'illusion de mouvement.

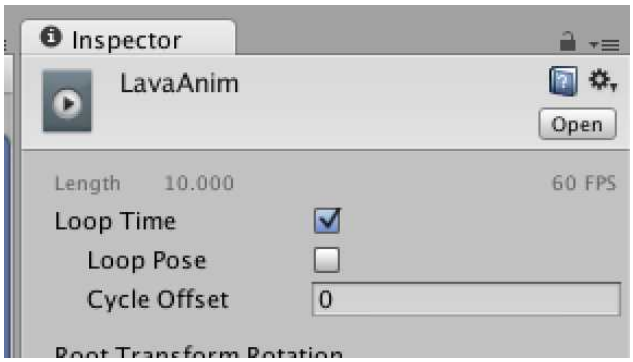
50.2. Entrez le nombre **300** comme numéro d'image.

50.3. Appuyez sur le bouton « Add Keyframe ».

50.4. À la droite du texte ***Position.y***, inscrivez la valeur **69** ou une autre valeur appropriée pour votre scène.

Fermez la fenêtre d'animation, vous avez complété votre clip.

Vous allez spécifier que le ***clip*** doit boucler.



51. Dans le panneau *Project*, sélectionnez le clip d'animation **Lava_Anim.**

51.1. Dans *l'Inspector*, cochez (si ce n'est pas déjà fait) ***Loop Time*.**

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Testez votre jeu.

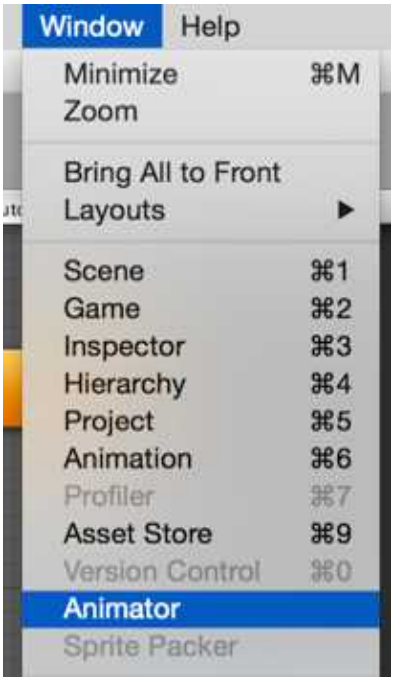
Conseil!

Si l'animation joue trop vite à votre goût, vous pouvez changer la vitesse de lecture dans la fenêtre *Animator*.

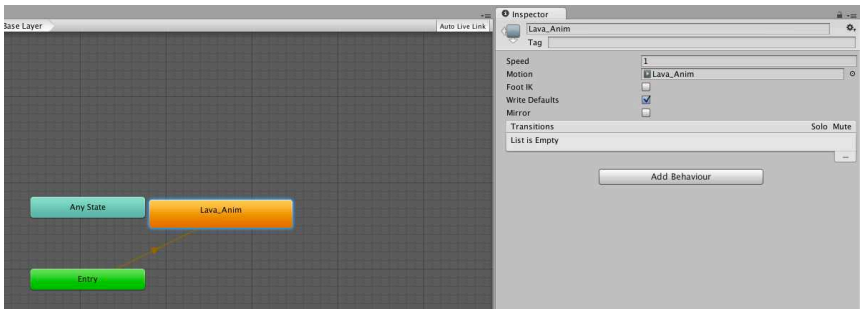
Retournez en mode édition.

52. Allez dans la barre de menus, puis cliquez sur :

Window | Animator



Dans cette fenêtre, vous pouvez changer la vitesse d'un clip. L'animation en orange représente le clip qui joue par défaut. Si vous sélectionnez un clip, vous avez quelques paramètres dont ***Speed***.



53. Ajustez la vitesse de lecture selon vos préférences.

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

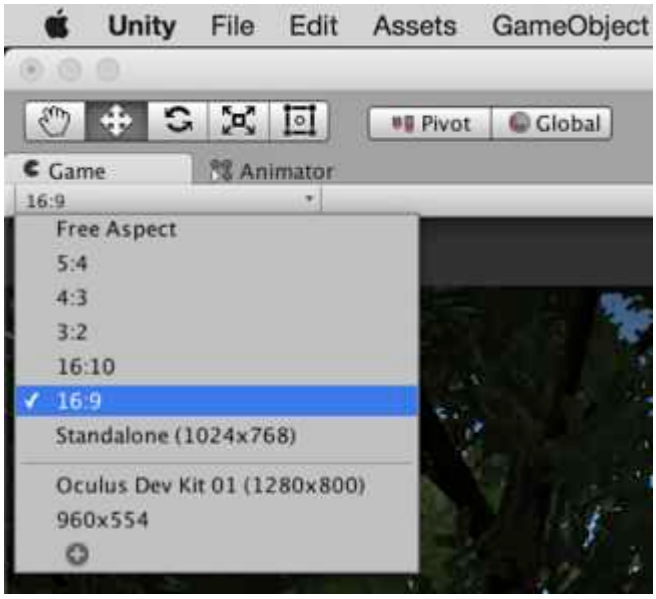
Testez votre jeu.

Retournez en mode édition.

Voici ce qui finalise la création de la lave du volcan. La prochaine section aura pour objectif de mettre des éléments visuels dans l'interface.

Création d'un radar

Unity® 4.0 introduisait *Mecanim* comme nouveau système d'animation. La version 5 ajoute un nouveau système d'interface utilisateur (***UI***). Vous allez explorer certains aspects de cet outil dans cette partie de l'exercice.



Afin d'obtenir les meilleurs résultats possibles avec le nouveau système d'***UI***, vous devez spécifier le ratio de l'écran à **16:9.**

Juste en-dessous de l'onglet du panneau *Game* (quand celui-ci est ouvert), vous avez une boîte de défilement avec différents aspects ratios.

1. Choisissez **16:9** parmi la liste.

Notez-bien!

Vous pouvez également ajouter une nouvelle résolution ou ratio en appuyant sur le ***(+)*** *au bas de cette liste.*

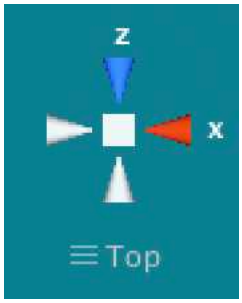
Préparation de la carte

Pour la prochaine étape, nous utiliserons un logiciel de traitement d'images comme ***Adobe Photoshop*** pour créer une carte topographique de l'île. Si vous n'avez pas de logiciel de traitement d'images, vous pouvez vous servir d'une capture d'écran et de Microsoft **Paint** sur *Windows* ou le logiciel **Aperçu** sur *Mac* afin de sauver votre texture.

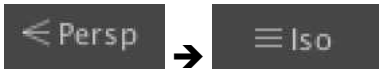
Notez-bien!

Il existe d'autres logiciels qui sont gratuits et qui peuvent vous servir. ***Gimp*** *est un logiciel Open-Source disponible pour toutes les plateformes (Windows, OS X, Linux) et il est très complet. Il existe également Autodesk* ***Pixlr*** *qui fonctionne dans un fureteur Web compatible avec* ***Adobe Flash****. Présentement, l'importance c'est de créer une représentation fidèle de la place sur un plan 2D.*

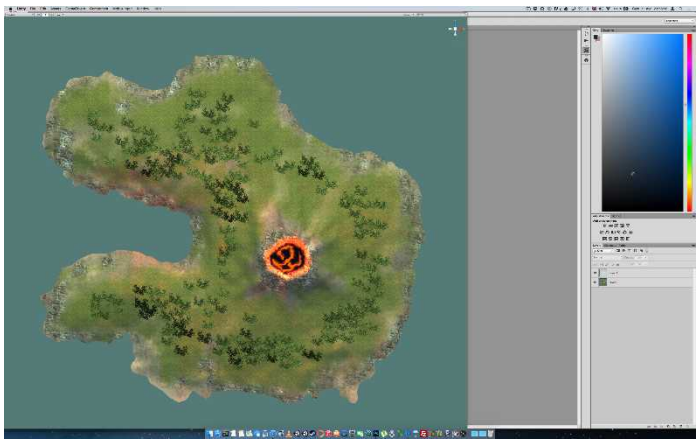
2. Dans la fenêtre *Scene*, appuyez sur l'axe **Y** du ***Gizmo*** afin de vous placer au-dessus de votre île.



2.1. Si vous n'obtenez pas le résultat attendu, cliquez sur les lignes en-dessous du ***Gizmo*** pour vous mettre en vision isométrique.

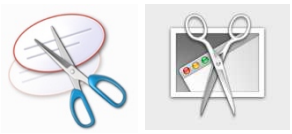


2.2. Cadrez votre île le mieux possible dans l'éditeur afin de pouvoir capturer la plus belle image souhaitée de votre terrain.



3. Faites une capture de l'écran :

• (**ALT+Imp.écran**) ou avec *l'**outil de capture*** sur ***Windows***
• (**⌘+⇧+3**) ou avec l'application ***Capture*** sur ***Mac***

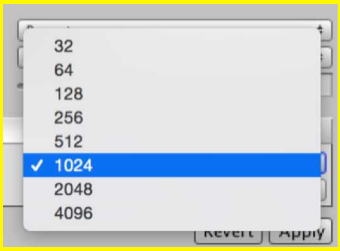


↘ *Les outils de capture qui sont fournis avec votre système d'exploitation offrent plusieurs options intéressantes pour capturer aisément un élément sur votre écran.*

Conseil!

On utilise les **puissances de 2** en informatique, car les ordinateurs utilisent le binaire (0 ou 1) pour opérer. C'est pourquoi on double la taille de l'image à chaque fois que l'on dépasse de 1 pixel l'une de ces frontières.

Il n'est pas nécessaire que les valeurs horizontales et verticales soient identiques, vous pouvez avoir des textures rectangulaires en **puissance de 2** :



(128x256, 1024x64, 512x1024, etc.)

4. ***Recadrez***, ***ajustez***, ***peignez*** et ***exportez*** votre carte avec un logiciel de traitement d'images afin d'obtenir le résultat désiré.

• Il est important que la carte couvre la totalité de votre île.
• Autant que possible, n'utilisez pas le format *.jpeg* pour vos textures, privilégiez plutôt les formats sans compressions destructives comme : (**.tga, .png,** ou même .**psd**).
• Il est important que vous exportiez votre image avec une dimension en puissance de **2** : **(32, 64, 128, 256, 512, 1024, etc.).**

Notez-bien!

Unity® réajuste les images en puissance de 2 si celles-ci ne sont pas correctement dimensionnées. Cependant, ce processus prend un certain temps de compression à chaque fois qu'un changement important survient. Aussi, redimensionner une image ajoute des artéfacts visuels. ***Un bon artiste utilise toujours des puissances de 2 pour ses textures.***

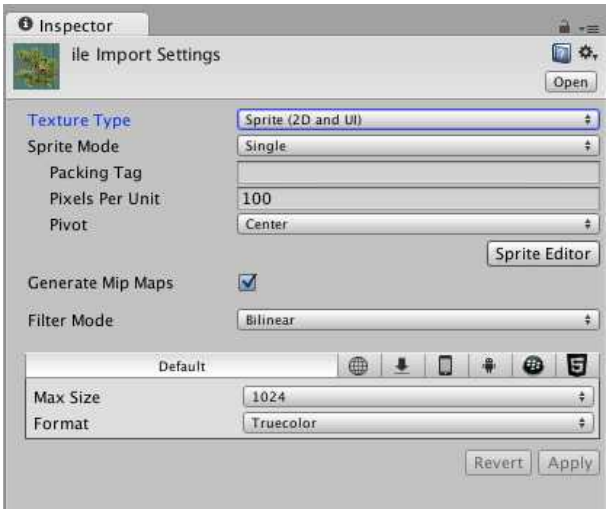
5. Importez votre carte dans votre projet.

↘ *Personnellement, j'utilise une texture de 1024x1024 au format (.tga) 16bit que j'ai créée avec Photoshop.*

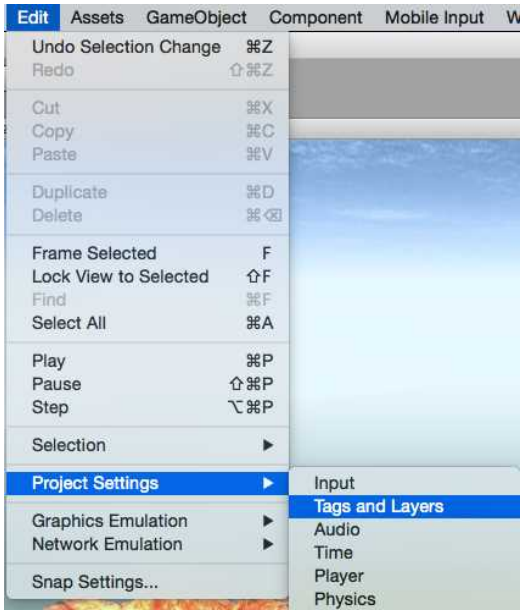
6. Sélectionnez votre image dans *Project*.

6.1. Changez les paramètres suivants dans *l'Inspector :*

6.1.1. ***Texture Type*** **:** Sprite (2D and UI)
6.1.2. ***Sprite Mode*****:** Single
6.1.3. ***Generate Mip Maps*****:** oui



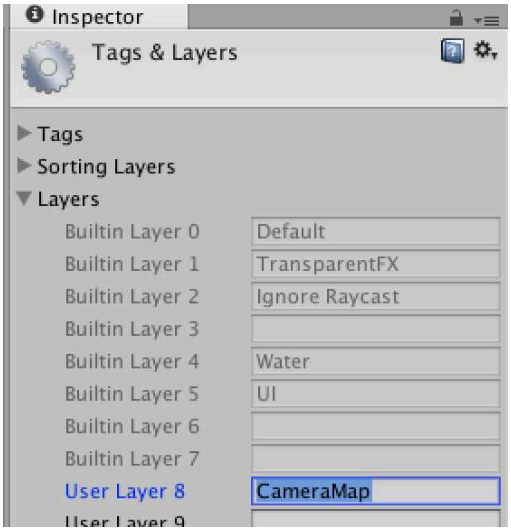
6.2. Appuyez sur « Apply ».



Vous avez besoin d'un calque (***layer***) pour isoler les éléments de la carte.

7. Dans la barre de menus, allez :

Edit | Project Settings | Tags and Layers

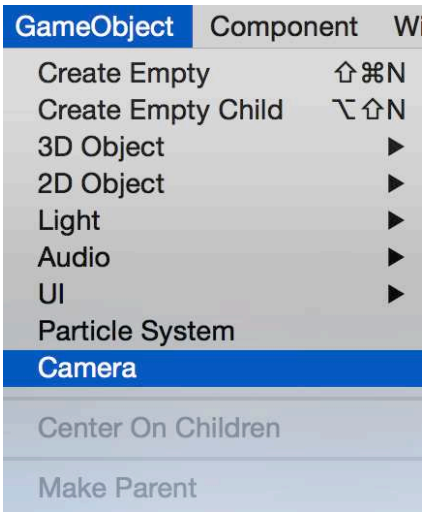


7.1. Dans la liste ***layers*** inscrivez **CameraMap** dans le champ ***User Layer 8.***

Vous allez créer une caméra pour votre interface. Cette caméra ne verra que les éléments qui se trouvent sur le ***layer*** **CameraMap.**

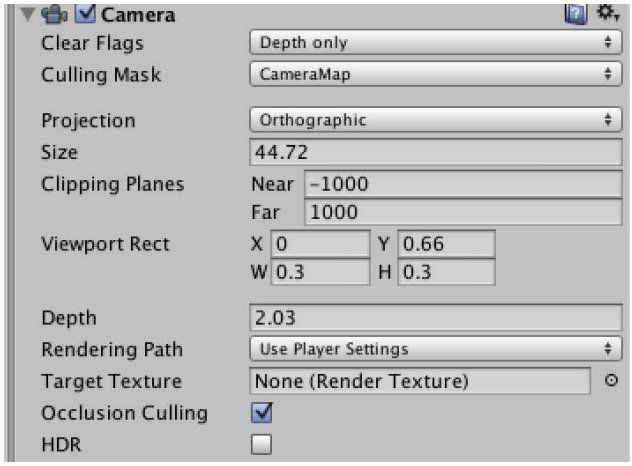
8. Dans la barre de menus, allez :

GameObject | Camera



8.1. Nommez-la **CameraMap.**

8.2. Dans *l'Inspector*, changez :



Transform

8.2.1. ***Rotation*** : **X** = 90 (afin qu'elle pointe vers le bas).

Camera

8.2.2. ***Clear Flags*** : Depth only

8.2.3. ***Projection*** : Orthographic

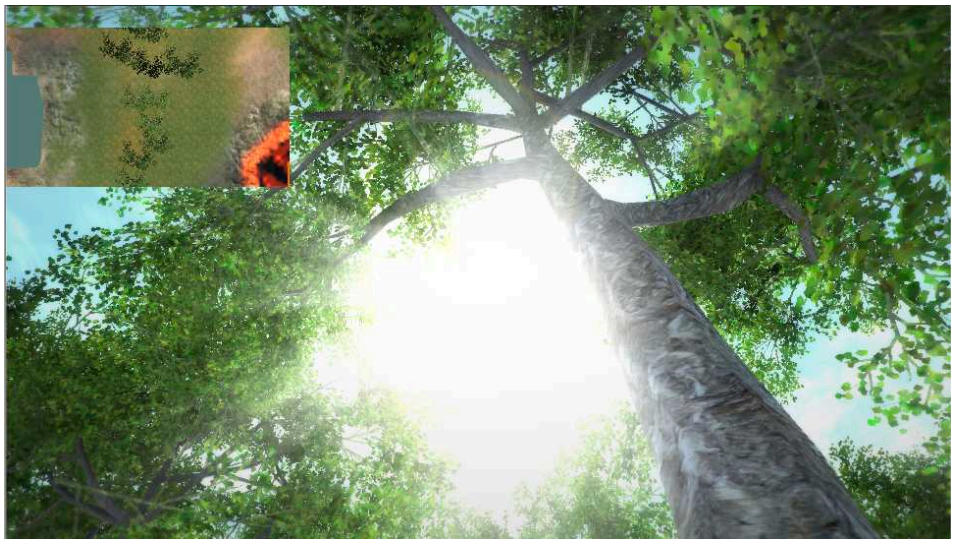
8.2.4. ***Size*** : 44.7

8.2.5. ***Clipping Planes*** : **Near**= -1000, **Far**= 1000

8.2.6. ***Viewport Rect*** : **X=** 0, **Y=** 0.66, **W=** 0.3, **H=** 0.3

8.2.7. ***Depth*** = 2

Vous devriez voir votre scène dans le coin en haut, à gauche de la fenêtre *Game.*



Vous allez changer ce que la caméra peut voir afin de la limiter à votre carte.

8.3. Toujours dans *l'Inspector*, changez :

Camera
8.3.1. ***Culling Mask*** : Décochez tout, sauf : **CameraMap.**

Votre carte va disparaître pour l'instant.

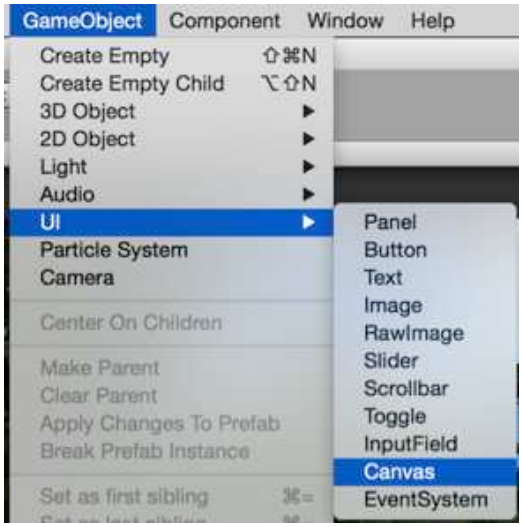
Afin d'éviter des messages d'avertissements inutiles :

8.4. Décochez le ***Component*** **Audio Listener** de cette caméra.

Pour la création de la carte, vous allez utiliser un canevas placé dans la scène, visible uniquement par la nouvelle caméra.

9. À partir de la barre de menus, allez dans :

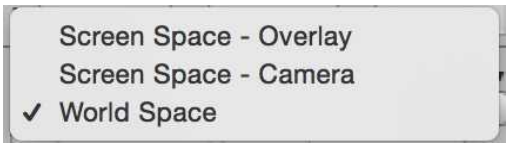
GameObject | UI | Canvas



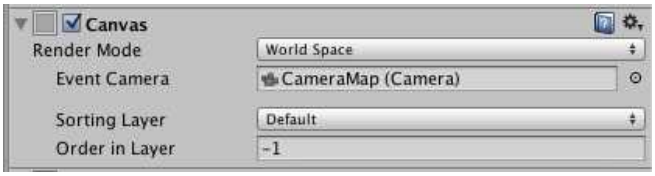
9.1. Nommez-le **CanevasMAP.**

Vous allez maintenant signaler à votre canevas que vous voulez qu'il occupe un espace physique dans votre scène.

9.2. Dans *l'Inspector,* changez les valeurs du ***Component*** :



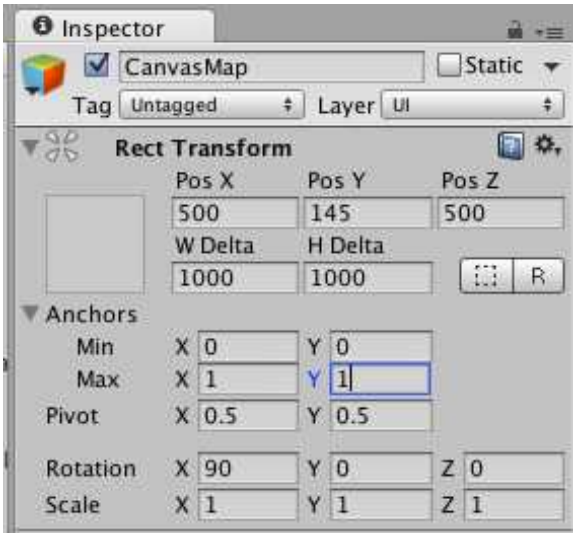
Canvas:
9.2.1. ***Render Mode*** : **World Space**
9.2.2. ***Event Camera*** : Glissez la caméra : **CameraMap.**
9.2.3. ***Order in Layer*** : -1



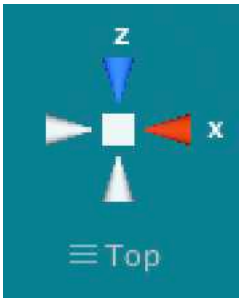
Rect Transform :
9.2.4. **Pos : X** = 500, **Y** = 145, **Z** = 500
Il est préférable de garder ce paramètre pour la fin.

9.2.5. ***Rotation*** **: X**= 90

9.2.6. ***W Delta*** = 1000, ***H Delta*** = 1000
9.2.7. ***Anchors Min*** **: X=** 0, **Y=** 0
9.2.8. ***Anchors Max*** **: X=** 1, **Y=** 1
9.2.9. ***Pivot*** **: X=** 0.5, **Y=** 0.5



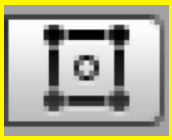
10. Dans le fenêtre *Scene*, appuyez sur l'axe **Y** du ***Gizmo*** afin de vous placer au-dessus de votre île.



10.1. Si vous n'obtenez pas le résultat attendu, cliquez sur les lignes en-dessous du ***Gizmo*** pour vous mettre en vision isométrique.

Conseil!

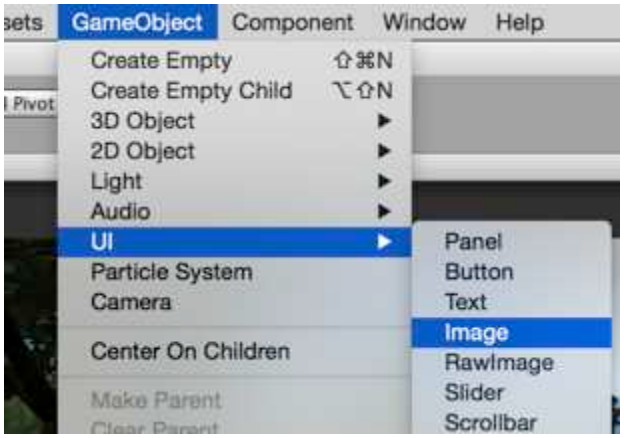
Si vous devez apporter des ajustements à votre canevas, utilisez **UI** qui se trouve dans la *barre d'outils* :



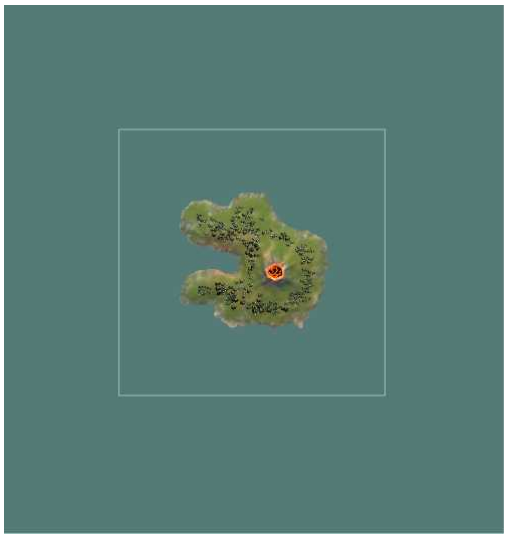
Maintenant, procédons à l'ajout de la carte 2D.

11. Dans la barre de menus, allez dans :

GameObject | UI | Image



Voici ce que vous devriez obtenir :

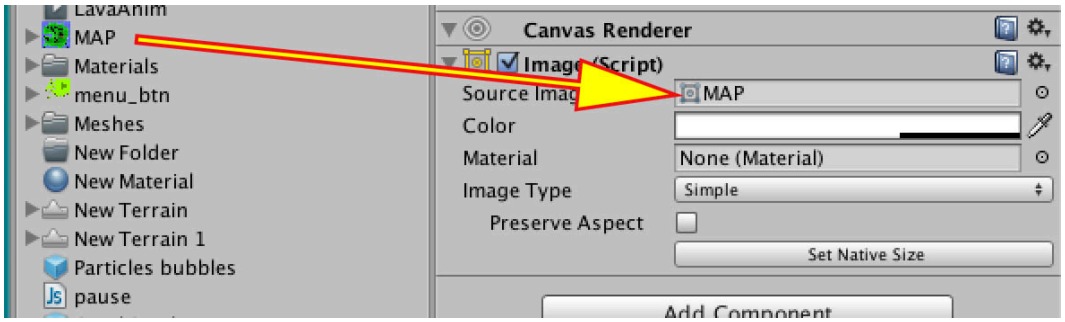


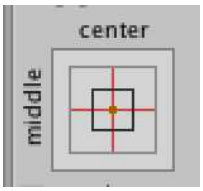
11.1. Nommez l'image **radar.**

11.2. Dans *l'Inspector,* changez les éléments suivants :

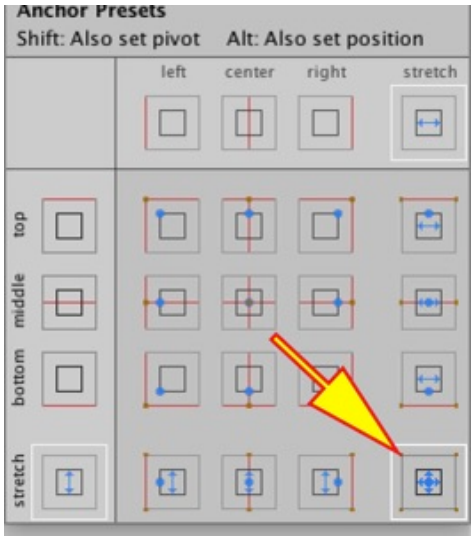
Image (Script)

11.2.1. ***Source Image*** : Glissez l'image de votre **carte 2D**.





11.3. Dans le haut de votre *Inspector,* localisez l'icône d'alignement et de la taille, puis changez :



Rect Transform

11.3.1. ***Anchor Presets*** : **Stretch** horizontal, **Stretch** vertical

11.3.2. ***Left*****=** 0, ***Top*****=** 0, ***Right*****=** 0, ***Bottom*****=** 0, ***Pos :*** **Z=** 0
(Il est préférable de garder ce paramètre pour la fin).

11.3.3. ***Anchors Min*** **: X=** 0, **Y** = 0
11.3.4. ***Anchors Max*** **: X** = 1, **Y** = 1
11.3.5. ***Pivot*** **: X=** 0.5, **Y=** 0.5

En détails...

Votre carte devrait couvrir le canevas, mais les petits ajustements sont présentement difficiles à régler à cause de l'opacité de l'image.

11.4. Dans *l'Inspector,* changez :

***Component* image (Script)**

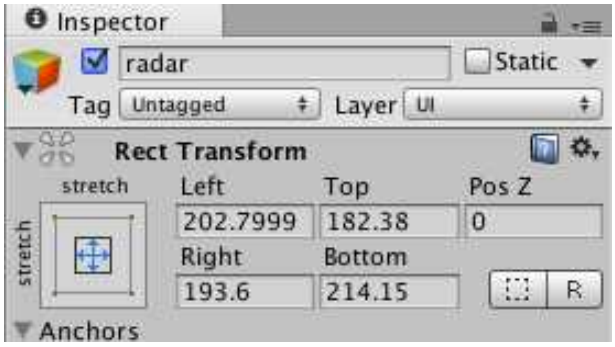
11.4.1. Cliquez sur le champ ***Color*** pour ouvrir les propriétés avancées de la couleur.

11.4.2. Changez la valeur du champ (**A**) pour une valeur entre **60-70**.

L'image devrait devenir semi-transparente et il sera alors plus facile d'ajuster les petits détails.

Rect Transform,

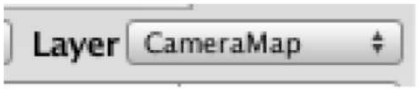
11.4.3. ***Left, Right, Top*** et ***Bottom*** : Ajustez les valeurs pour que la carte couvre parfaitement l'île.



Conseil!

Il est important d'être le plus précis possible afin que la carte soit fidèle à la topologie du monde.

Vous allez changer le ***layer*** de l'image afin que seule la caméra de la carte puisse la voir.



11.5.Changez le ***layer*** de l'image **CanvasMap** de ***UI*** pour ***CameraMap.***

La carte devrait redevenir visible dans la fenêtre *Game*.

Conseil!

Réajustez la transparence selon vos goûts comme vous l'avez fait précédemment.

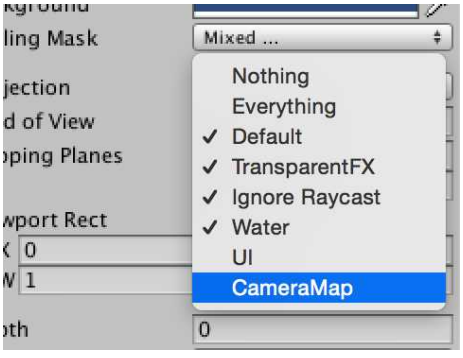
Le problème que se présente alors, c'est que la caméra du joueur voit la carte flotter dans le ciel.

12. Dans le panneau *Hierarchy,* sélectionnez l'objet : **Main Camera** qui se trouve à l'intérieur du **First Person Controller**.

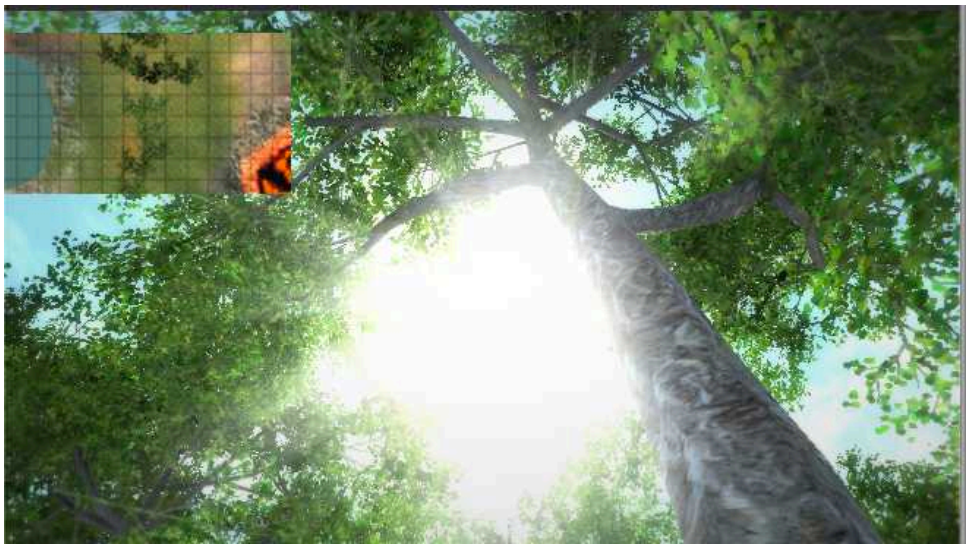
12.1. Dans *l'Inspector,*

Camera

12.1.1. ***Culling Mask*** : Décochez **CameraMap**.



La carte devrait être visible seulement de la caméra.

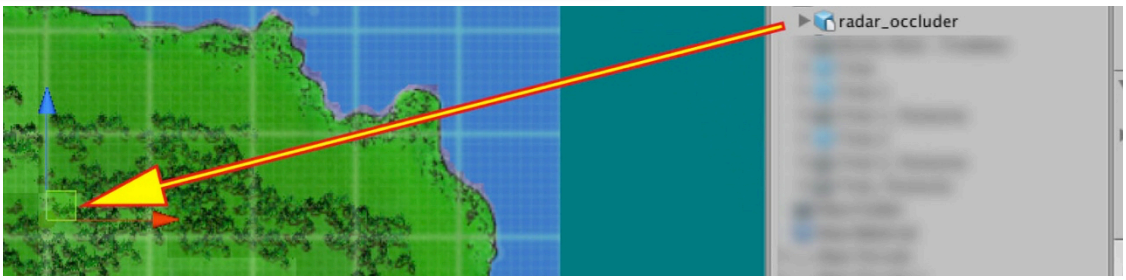


Notez-bien!

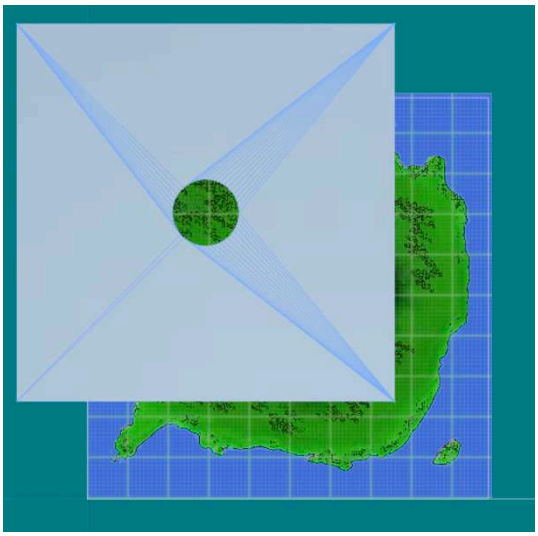
Présentement, l'affichage de la carte est imparfait. Il est évident qu'un masque en forme de cercle serait beaucoup plus approprié. Malheureusement, le système d'interface d'Unity® 5.0 est présentement limité quand il est question de masques.

*C'est pourquoi, il faut utiliser un **Shader** personnalisé et un **Mesh** avec un trou au centre pour créer notre effet. Ces éléments proviennent directement de l'exemple **Boot Camp** par Unity qui était distribué gratuitement sur **l'Asset Store**. J'ai rassemblé dans le **package** le **Shader** nécessaire.*

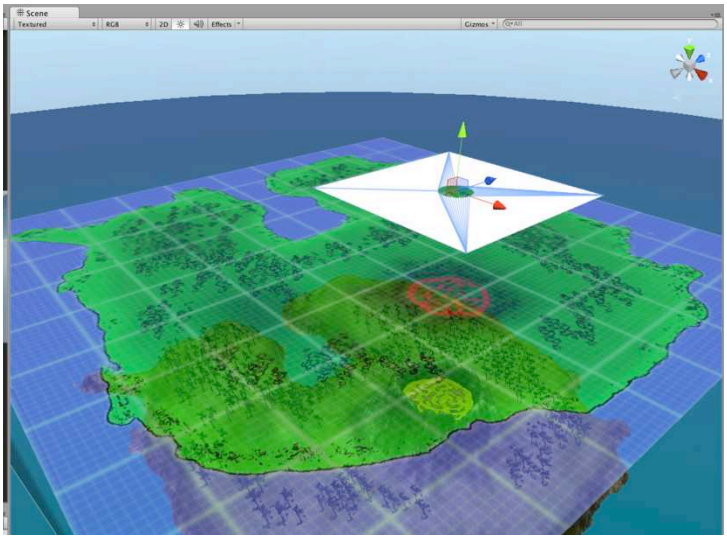
13. À partir de *Project,* glissez l'objet **radar_occluder** qui se trouve dans le répertoire **FBX** vers votre scène.



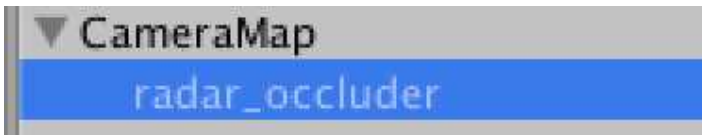
14. Avec l'outil de ***redimensionnement***, agrandissez **radar_occluder** afin que le trou couvre une portion réduite du terrain.



15. Réajustez ensuite la hauteur de **radar_occluder** au-dessus de la carte, comme ceci :



16. Dans le panneau *Hierarchy*, mettez **radar_occluder** enfant de **CameraMap.**



En détails...

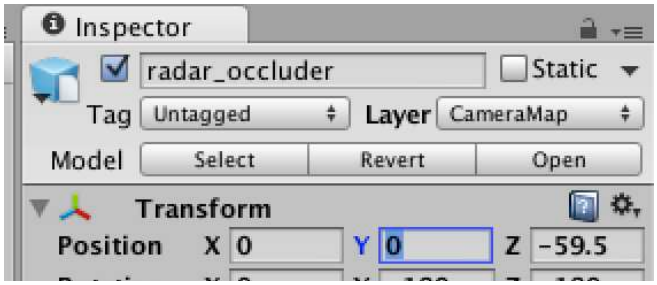
R**adar_occluder** est maintenant enfant de la caméra, ses coordonnées sont relatives à son parent.

17. Sélectionnez **radar_occluder** dans votre *Hierarchy,* puis :

17.1. Dans *l'Inspector,* changez :

17.1.1. ***Layer*** : CameraMap

Transform
17.1.2. ***Position*** **: X=** 0, **Y=** 0, **Z=** -59.5



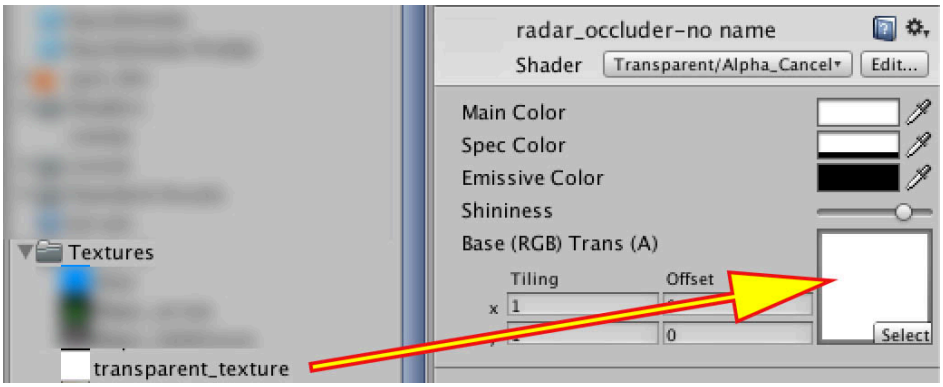
Vous devriez voir le trou sur la carte, il reste à ajouter le ***Shader*** d'occlusion pour cacher ce qui apparaît en blanc.

18. Dans le panneau *Project,* sélectionnez **radar_occluder-no name** qui se trouve dans le répertoire **Material.**

18.1. Dans *l'Inspector*, changez :

18.1.1. ***Shader*** **:** changez **Default** pour **Transparent/Alpha_Cancel**

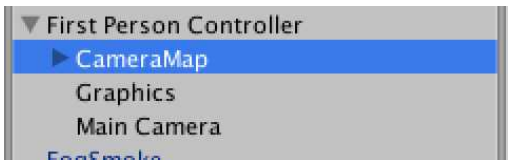
18.1.2. ***Base (RGB) Trans (A)*** **:** Glissez la texture **transparent_texture** qui se trouve dans votre répertoire 0 **Textures.**



Vous devriez obtenir ce résultat :

En détails...

Vous allez attacher **cameraMAP** au **First Person Controller** pour que celle-ci suive les déplacements du joueur.

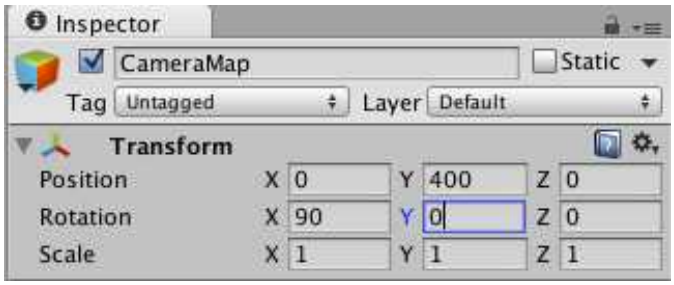


19. Dans le panneau *Hierarchy*, mettez **CameraMap** enfant de **First Person Controller.**

20. Sélectionnez **CameraMap.**

20.1. Dans *l'Inspector,* changez :

Transform
20.1.1. ***Position*** **: X=** 0**, Y=** 400**, Z=** 0
20.1.2. ***Rotation*** **: Y=** 0

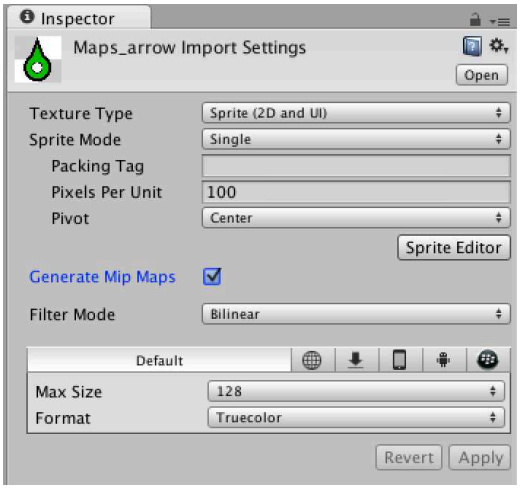


En détails...

Comme la **caméra et l'occlusion** sont enfants du joueur, la rotation ainsi que la position de la carte seront automatiquement synchronisées.

Vous allez ajouter un curseur pour indiquer la position du joueur sur la carte.

21. À partir du panneau *Project*, sélectionnez l'image **Maps_arrow** du répertoire **Textures.**

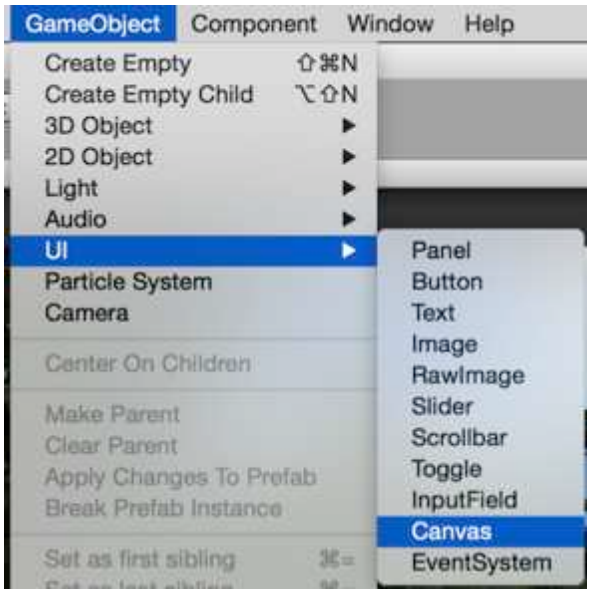


21.1. Dans *l'Inspector,* changez les propriétés :

21.1.1. ***Texture Type*** : Sprite (2D and UI)
21.1.2. ***Generate Mip Maps*** : oui
21.1.3. ***Format*** : Truecolor

21.2. Appuyez sur « Apply ».

Vous allez ajouter un nouveau canevas, uniquement pour ce curseur.



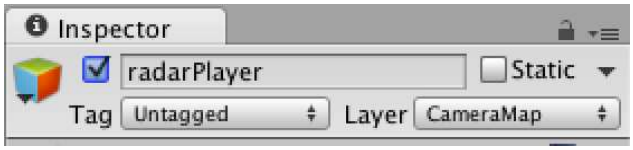
22. Allez dans la barre de menus :

GameObject | UI | Canvas

22.1. Nommez-le **radarPlayer.**

22.2. Dans *l'Inspector*, changez :

22.2.1. ***Layer*** pour **CameraMap.**

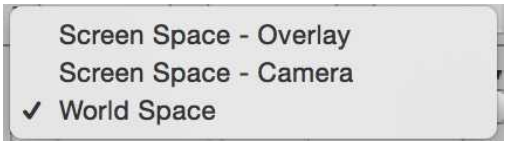


23. Dans votre *Hierarchy*, mettez **radarPlayer** enfant de **First Person Controller.**

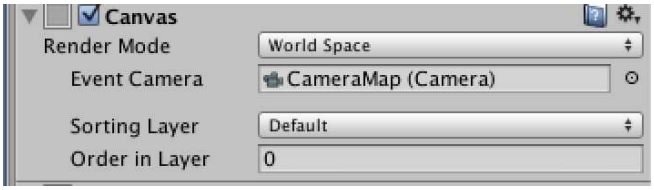
Vous allez indiquer à votre canevas que l'on veut qu'il occupe un espace physique dans la scène.

24. Sélectionnez **radarPlayer**, si ce n'est pas déjà fait.

24.1. Dans *l'Inspector,* changez :

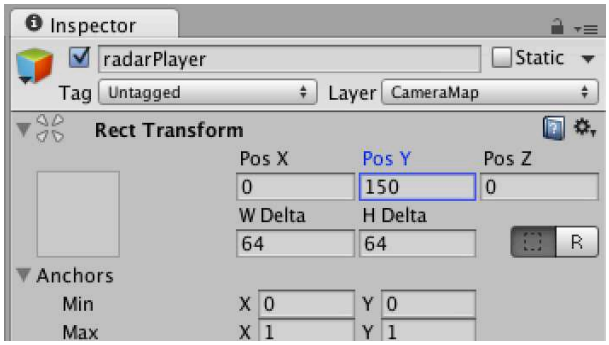


Canvas
24.1.1. ***Render Mode*** : **World Space**.



24.1.2. ***Event Camera,*** glissez sur **CameraMap (camera).**

Rect Transform
24.1.3. ***Pos*** **: X=** 0, **Y=** 150, **Z=** 0
(Il est préférable de garder ce paramètre pour la fin).

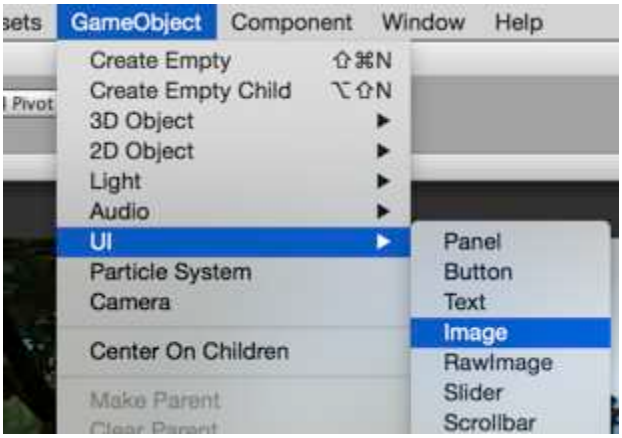


24.1.4. ***W Delta*** = 64, ***H Delta*** = 64
24.1.5. ***Anchors Min*** **: X**= 0, **Y=** 0
24.1.6. ***Anchors Max*** **: X=** 1, **Y=** 1
24.1.7. ***Pivot*** **: X =** 0.5, **Y** = 0.5
24.1.8. ***Rotation*** **: X=** 90, **Y=** 0, **Z=** 0

Le canevas est maintenant centré sur le joueur.

25. Dans la barre de menus, allez dans :

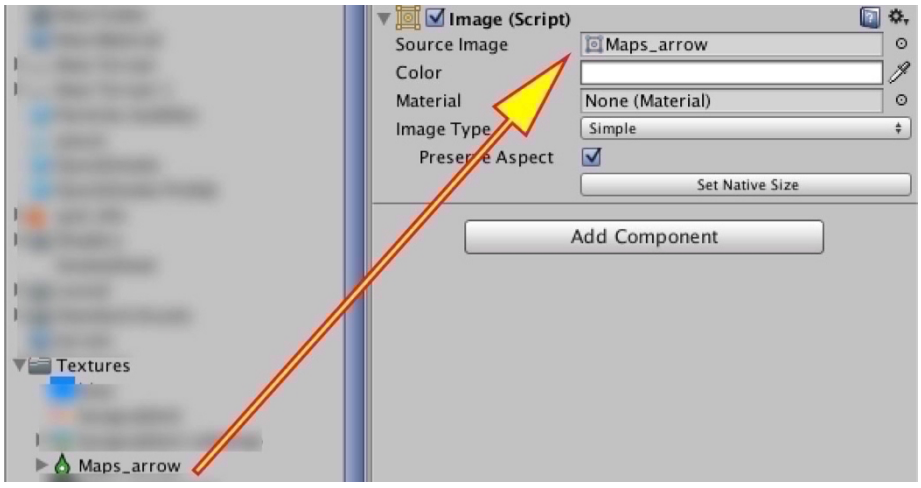
GameObject | UI | Image

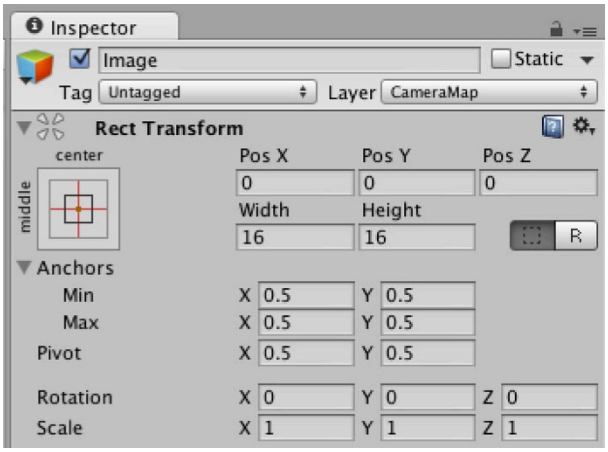


25.1. Renommez l'image **cursor.**

25.2. Dans *l'Inspector*, changez :

Image (Script)
25.2.1. ***Layer***: **CameraMap**
25.2.2. ***Source Image,*** glissez l'image sur **Maps_arrow.**





Rect Transform:
25.2.3. ***Pos*** **: X=** 0, **Y=** 0, **Z=** 0
25.2.4. ***W Delta*** = 16, ***H Delta*** = 16
25.2.5. ***Anchors Min*** **: X=** 0.5, **Y=** 0.5
25.2.6. ***Anchors Max*** **: X=** 0.5, **Y=** 0.5
25.2.7. ***Pivot*** **: X=** 0.5, **Y=** 0.5

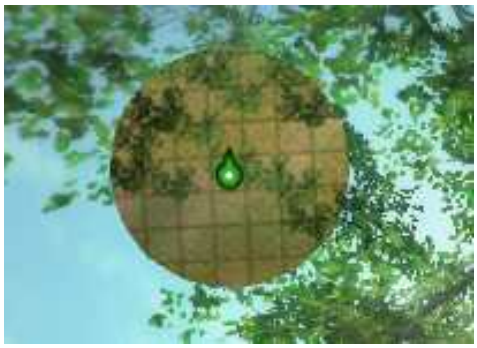
Le curseur est maintenant visible à l'écran.

Sauvez votre scène !

Testez votre jeu.

Votre radar est maintenant terminé.

Retournez en mode édition.



Chapitre 9 - Le terrain : partie 3

Matière couverte dans ce chapitre

- **Ajout de filtres de couleurs**
- **Raffinement en-dessous de l'eau**
- **Ajout d'un mort par le volcan**
- **Ajout de dommages corporels**
- **Création d'un menu de pause**

Nous complèterons le terrain des deux chapitres précédents, en y ajoutant des filtres de couleurs pour créer des effets spéciaux. Il y en aura sous l'eau, dans le volcan et lorsque le personnage agonisera. Finalement, nous mettrons un menu de pause, celui-ci arrêtera le déroulement du jeu et permettra au joueur de le quitter ou de le reprendre.

Atelier pratique

Effets spéciaux et menu

Débutez en ouvrant le projet **Terrain** des chapitres précédents.

Ouvrez la scène **Game.**

Interfaces utilisateurs (UI) : Ajout de filtres de couleurs

Dans cette partie de l'exercice, nous allons ajouter 3 filtres de couleurs pour créer différents effets spéciaux. Un filtre bleu sera nécessaire quand le joueur est sous l'eau, un filtre orange lorsqu'il se jette dans le volcan et un filtre rouge quand il chute de trop haut.

Avant de créer l'interface, nous devons définir un ***Canvas***. Tout comme le canevas pour un peintre, celui-ci est une surface sur laquelle on ajoute des éléments. Nous pouvons également utiliser plusieurs canevas dans une même scène. Il est possible d'animer le canevas ou les éléments qu'il contient et spécifier comment celui-ci s'ajuste aux différentes résolutions et aspects ratios des écrans.

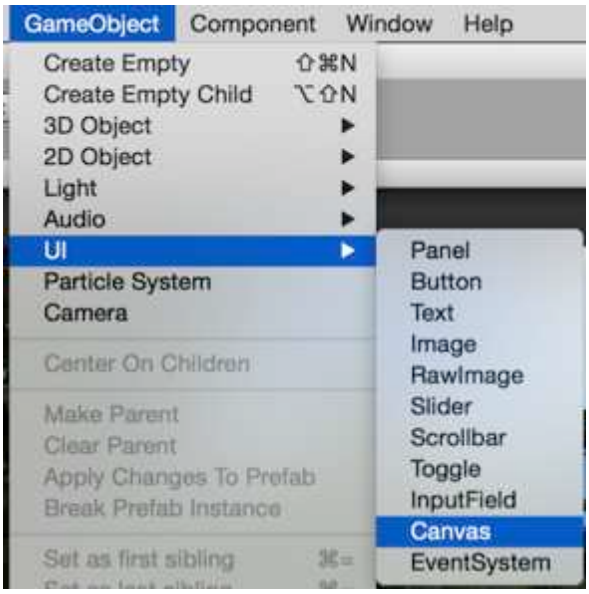
Pour que l'effet soit réussi, nous avons besoin d'un canevas qui va s'ajuster automatiquement aux extrémités de l'écran.

1. Dans la barre de menus, allez dans :

GameObject | UI | Canvas

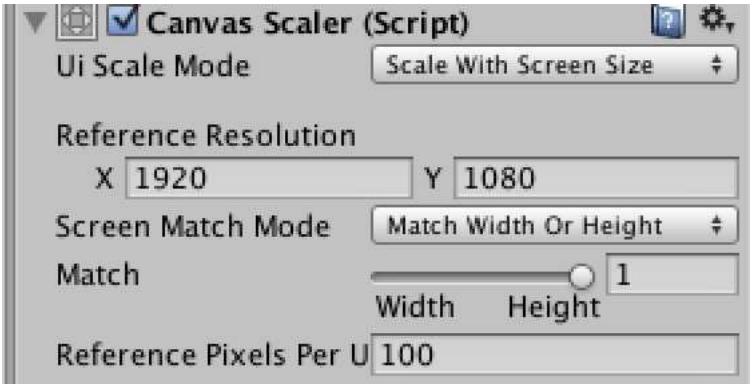
2. Sélectionnez **Canvas** dans *Hierarchy.*

2.1. Renommez-le **CanvasUI.**



2.2. Dans *l'Inspector*, changez les paramètres :

Canvas
2.2.1. ***Render Mode*** : Screen Space - Overlay
Canvas Scaler
2.2.2. ***UI Scale Mode*** : Scale with Screen Size
2.2.3. ***Reference Resolution*** **: X=** 1920, **Y=** 1080
2.2.4. ***Screen Match Mode*** : Match Width Or Height
2.2.5. ***Match*** : **Height =** 1
2.2.6. ***Reference Pixel Per Unit*** : 100



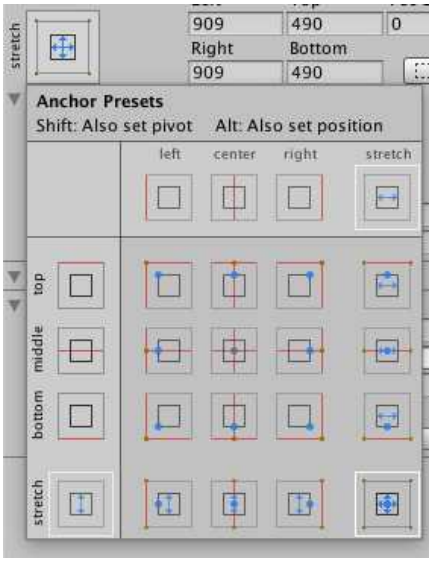
En spécifiant que nous utilisons le ***Screen Match Mode*** avec la hauteur (***Height***) égale à **1** ou 100%, nous définissons que la hauteur sert de référence pour le redimensionnement.

Préparation des effets

3. Dans votre *Hierarchy*, sélectionnez **CanvasUI**.

3.1. À partir de la barre de menus :

GameObject | UI | Image



4. Sélectionnez cette image.

4.1. Nommez-la **underwaterFx.**

4.2. Dans *l'Inspector***,** changez les paramètres:

Rect Transform
4.2.1. ***Anchor Presets*** (**Stretch** horizontal et **Stretch** vertical).
4.2.2. ***Left*****=** 0, ***Right*****=** 0, ***Top*****=** 0, ***Bottom*****=** 0, ***pos*** **Z**= 0
4.2.3. ***Anchors Min*** **: X=** 0, **Y=** 0
4.2.4. ***Anchors Max*** **: X=**1, **Y=** 1
4.2.5. ***Pivot*** **: X=** 0.5, **Y=** 0.5

Image (Script)
4.2.6. ***Source Image*** : Aucune
4.2.7. ***Color* :** Bleu, semi-transparent
4.2.8. ***Material*** : Aucun
4.2.9. ***Preserve Aspect*** *:* Non

4.3. Décochez le ***Component*** **Image (Script)**, pour rendre l'image invisible à la caméra.

Vous allez refaire les mêmes étapes pour l'effet de lave et de dommage.

5. Dans votre *Hierarchy*, sélectionnez **CanvasUI**.

5.1. Puis, allez dans la barre de menus :

GameObject | UI | Image

6. Sélectionnez cette image

6.1. Nommez l'image **lavaFx.**

6.2. Dans *l'Inspector*, changez les paramètres suivants :

Rect Transform
6.2.1. ***Anchors Min* : X=** 0, **Y=** 0
6.2.2. ***Anchors Max* : X=** 1, **Y=** 1
6.2.3. ***Pivot* : X=** 0.5, **Y**= 0.5
6.2.4. ***Left*=** 0, ***Right*=** 0, ***Top*=** 0, ***Bottom*=** 0, ***pos* Z**= 0

Image (Script)
6.2.5. ***Source Image*** : Aucune
6.2.6. ***Color* :** Orange, semi-transparent
6.2.7. ***Material*** : Aucun
6.2.8. ***Preserve Aspect*** : Non

6.3. Décochez le ***Component* Image (Script)** pour rendre l'image invisible à la caméra.

7. Dans votre *Hierarchy*, sélectionnez **CanvasUI.**

8. Puis, allez dans la barre de menus :

GameObject | UI | Image

9. Sélectionnez cette image.

9.1. Nommez l'image **damageFx.**

9.2. Dans *l'Inspector*, changez les paramètres suivants :

Rect Transform
9.2.1. ***Anchors Min* : X=** 0, **Y** = 0
9.2.2. ***Anchors Max* : X=** 1, **Y** = 1
9.2.3. ***Pivot* : X=** 0.5, **Y**= 0.5
9.2.4. ***Left*= 0, *Right*= 0, *Top*= 0, *Bottom*= 0, *Pos* Z=** 0

Image (Script)
9.2.5. ***Source Image*** : Aucune
9.2.6. ***Color* :** Rouge, semi-transparent
9.2.7. ***Material*** : Aucun
9.2.8. ***Preserve Aspect*** : Non

9.3. Décochez le ***Component* Image (Script)** pour rendre l'image invisible à la caméra.

Vous avez les filtres nécessaires, il faut désormais ajouter les scripts pour les rendre fonctionnels.

Sous l'eau

Vous allez créer un fichier ***JavaScript*** pour l'eau.

1. Faites un clic secondaire dans *Project,* puis :

➢ **Create > Javascript**

1.1. Nommez-le **underwater.**

Ouvrez le script « underwater.js ».

Ajoutez le code suivant :

```
var underwaterEffect:UI.Image;
var waterObj:Transform;
var diveSound:AudioClip;
var ObjectToHide:GameObject[];

private var isUnder:boolean = false;

function Update () {
  var audioS = GetComponent.<AudioSource>();
  if (transform.position.y < waterObj.position.y && ! isUnder){
     underwaterEffect.enabled = true;
     audioS.clip = diveSound;
     audioS.Play();
     isUnder = true;
     for each (GameObject in ObjectToHide)
        GameObject.active = false;
  }
  if (transform.position.y > waterObj.position.y && isUnder){
     underwaterEffect.enabled = false;
     isUnder = false;
     for each (GameObject in ObjectToHide)
        GameObject.active = true;
  }
}
```

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Le script (**transform.position.y**) nous permet de voir si le joueur est en-dessous (**<**) du niveau de l'eau (**waterObj.position.y**), affiche le filtre (**underwaterEffect.enabled**) et produit du son (**audio.Play**) si la condition est vraie. Il retire les effets, si la condition est fausse. La variable (**isUnder**) conserve l'état du joueur.

2. Glissez ce script sur **Main Camera** qui se trouve dans **First person Controller**.

Vous devez également utiliser ***AudioSource*** sur cette caméra afin que le son de plongeon se produise correctement.

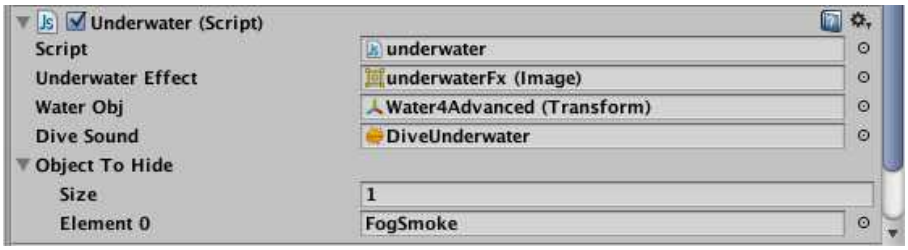
3. Sélectionnez **Main Camera** qui se trouve dans **First Person Controller.**

3.1. Dans *l'Inspector,* appuyez sur le bouton « Add Component », puis :

Audio | AudioSource

3.2. Toujours dans *l'Inspector*, changez :

Underwater (Script)
3.2.1. ***Underwater Effect*** : glissez l'effet **underwaterFx** de votre *Hierarchy*.
3.2.2. ***Water Obj*** : glissez l'objet **Water4Advanced** de votre *Hierarchy*.
3.2.3. ***Dive Sound*** : glissez le son **DiveUnderwater** de *Project*.
3.2.4. ***Object To Hide*** : **Size=**1
• ***Element 0*** : glissez l'émetteur de particules **FogSmoke** de votre *Hierarchy.*



En détails...

La raison pour laquelle on met **FogSmoke** dans une liste d'éléments à cacher c'est parce que ***Shader*** est incompatible avec le ***Shader*** d'eau. Pour régler le problème quand le joueur est sous l'eau, il faut cacher les particules.

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Testez votre application.

Vous devriez entendre un son de plongeon quand le joueur entre sous l'eau. Les particules de fumée devraient disparaître sous l'eau et redevenir visibles quand le joueur sort.

Retournez en mode édition.

Vous allez poursuivre en ajoutant un décès à cause de la lave.

Une petite baignade dans le volcan

Qu'arrive-t-il lorsqu'on se jette dans le cratère du volcan? Pour l'instant, rien! C'est pourquoi, nous allons créer un effet d'agonie, pour le joueur.

1. Sélectionnez **Main Camera** qui se trouve dans **First Person Controller.**

1.1. Dans *l'Inspector,* appuyez sur le bouton « Add Component », puis :

Image Effects | Blur | Motion Blur

1.2. Toujours dans *l'Inspector*, changez les valeurs suivantes :

Motion Blur (Script)
1.2.1. **Blur Amount** : 0.92
1.2.2. **Extra Blur** : Oui

1.3. Désactivez le ***Component,*** en le décochant :

Vous avez également besoin d'un **interrupteur** pour déclencher la mort du personnage dans le volcan.

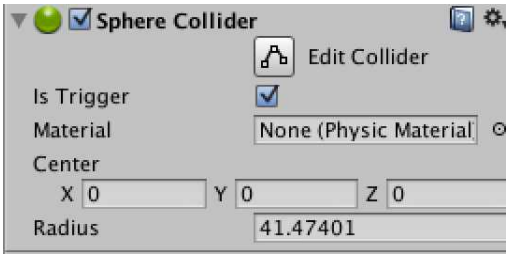
2. Allez dans la barre de menus, puis :

GameObject | 3D Object | Sphere

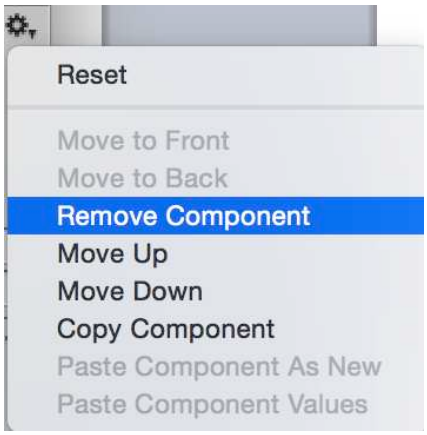
2.1. Nommez-la **LavaTrigger.**

2.2. Positionnez votre **sphère** au centre du volcan et changez :

Sphere Collider
2.2.1. ***Is Trigger*** : Oui
2.2.2. ***Radius*** : Ajustez-le pour qu'il couvre l'intérieur du volcan.



2.3. Retirez le component ***Mesh Renderer*** en appuyant sur **l'engrenage** dans le coin en haut à droite, puis appuyez sur « Remove Component ».



Notez-bien!

Contrairement aux colliders, les Triggers ne bloquent pas le joueur, lors d'une collision, c'est pourquoi nous les utilisons pour le volcan.

Afin que le script du volcan fonctionne, vous allez placer un marqueur (***Tag***) sur **First Person Controller.**

3. Sélectionnez **First Person Controller.**

3.1. Dans *l'Inspector*, changez ***Tag*** pour **Player**.

Pour le script qui va gérer ce ***Trigger*** :

4. Faites un clic secondaire dans le panneau *Project,* puis :

➢ **Create > Javascript**

4.1. Nommez le script **lava.**

4.2. Placez ce script sur le Trigger **LavaTrigger.**

Ouvrez le script « lava.js ».

Ajoutez le code suivant :

```
var lavaEffect:UI.Image;
var screamInPain:AudioClip;
var FPMainCamera:Camera;
var LavaComponent:UnityStandardAssets.ImageEffects.MotionBlur;

function OnTriggerEnter(col:Collider){
  var audioS = FPMainCamera.GetComponent.<AudioSource>();
  if (col.gameObject.tag == "Player"){
     audioS.clip = screamInPain;
     LavaComponent.enabled = true;
     audioS.Play();
     lavaEffect.enabled = true;
     yield WaitForSeconds (audioS.clip.length);
     Application.LoadLevel(Application.loadedLevel);
  }
}
```

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Ce script détecte la collision avec les différents objets (**OnTriggerEnter**). Si un objet porte le ***Tag*** **Player** (**col.gameObject.tag == "Player"**), alors on entend un son (**audio.Play**) d'agonie. On active l'effet de caméra et on affiche le filtre de couleur (**lavaEffect.enabled**). Après une pause (**yield WaitForSeconds**), on redémarre le niveau (**Application.LoadLevel**).

5. Dans votre *Hierarchy*, sélectionnez l'objet **LavaTrigger.**

5.1. Puis dans *l'Inspector*, changez :

Lava (Script)
5.1.1. ***Lava Effect*** : glissez l'image **lavaFx** de votre *Hierarchy.*
5.1.2. ***Scream In Pain*** : glissez le son **scream_in_pain** de *Project*.
5.1.3. ***FP Main Camera*** : glissez **Main Camera** de votre *Hierarchy.*
5.1.4. ***Lava Component*** : glissez **Main Camera** à nouveau.

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Testez le jeu : Jetez-vous dans le volcan…

AHHHHHHHHHHhhhhhhhhhhh!!!!!!!.....

Retournez en mode édition.

Notez-bien!

*Il est possible qu'il y ait un effet secondaire lorsqu'on redémarre l'**Application.LoadLevel()**. Si la scène est complètement assombrie comme dans l'exemple ci-dessous. Voici comment régler le problème.*

6. Dans la barre de menus, allez dans :

Window | Lighting

6.1. Dans ce panneau,

6.1.1. Décochez ***Continuous Baking***, car il cause un problème.

6.1.2. Appuyez sur le bouton « Build ».

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Testez le jeu

Il faut se jeter dans le volcan à nouveau.

Le problème d'éclairage devrait être réglé.

Retournez en mode édition.

Tant qu'à être masochiste, nous allons mettre en place un dernier effet. Les jambes vont craquer quand le joueur va tomber de trop haut!

C'est dur sur les genoux!

Ensuite, il faut créer un mouvement de tremblement sur la **caméra,** quand un impact survient.

Afin de ne pas affecter l'orientation de **Main Camera** du **First Person Controller**, vous allez créer le mouvement sur un ***GameObject*** parent à cette caméra.

1. Allez dans le menu :

GameObject | Create Empty

1.1. Nommez cet objet **Head.**

Afin que l'animation fonctionne, vous devez placer ce ***GameObject*** aux mêmes coordonnées que la caméra.

2. Mettez l'objet **Head** enfant de **Main Camera** du **First Person Controller**.

2.1. Dans *l'Inspector*, changez les propriétés de **Head** :

Transform
2.1.1. ***Position*** **: X=** 0**, Y=** 0**, Z=** 0
2.1.2. ***Rotation*** **: X=** 0**, Y=** 0**, Z** = 0

3. Mettez **Head** enfant du **First Person Controller.**

4. Pour terminer, mettez **Main Camera** enfant de **Head**.

Afin de créer une animation, vous devez ajouter un ***Animator*** sur **Head**.

En détails...

Cette petite gymnastique vous aura permis de créer un ***parent*** à **Main Camera** sans affecter son pivot.

5. Sélectionnez l'objet **Head** dans votre *Hierarchy.*

5.1. Dans *l'Inspector,* appuyez sur le bouton « Add Component », puis :

Miscellaneous | Animator

6. Ouvrez la fenêtre d'animation (**CTRL+6**) ou (**⌘+6**) sur mac ou à partir de la barre de menus :

Window | Animation

7. En appuyant sur le champ en-dessous des boutons d'animation, créez-en un nouveau clip.

7.1. Appuyez sur « [Create new clip] ».

7.2. Nommez le clip **damage.**

Vous devez définir ce que vous voulez animer. Dans ce cas-ci, c'est la ***rotation*** de l'objet **Head.**

8. Dans la fenêtre d'animation, appuyez sur « Add Curve », puis :

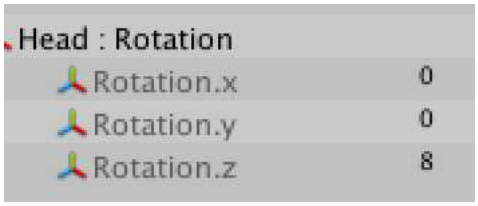
Transform | Rotation

8.1. Appuyez ensuite sur le **(+)** au bout de la ligne pour ajouter ce paramètre dans l'animation.

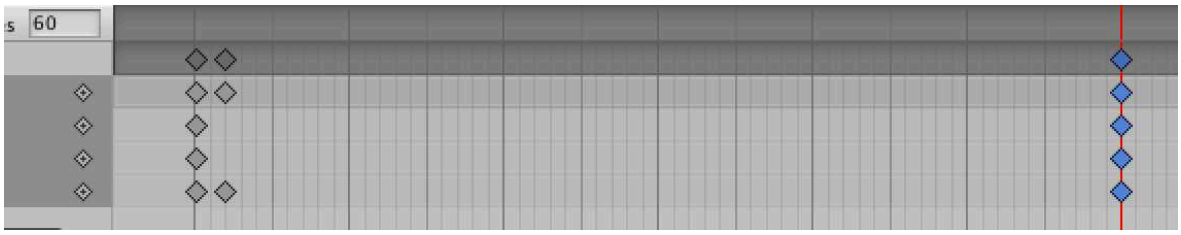
8.2. Dans la barre d'outils de la fenêtre d'animation, entrez le chiffre **2** dans la case du numéro d'image.

8.3. Dans les propriétés de ***Head : Rotation,*** changez :

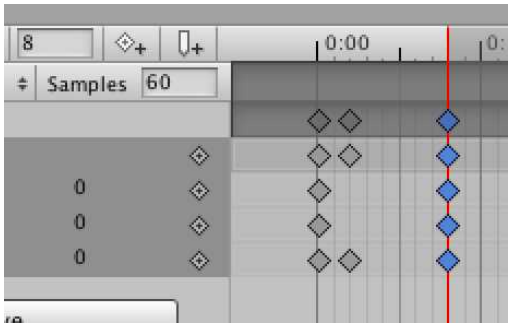
8.3.1. **Rotation.z= 8**



9. Sélectionnez les **images clés** qui sont à la fin de la ***Timeline.***

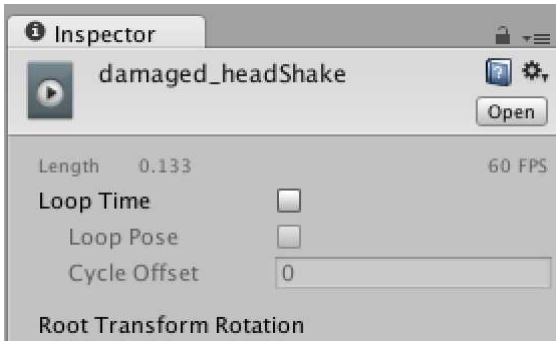


9.1. Glissez ces clés à l'image **8 (temps 0:08)** pour réduire la durée du clip.



10. Fermez la fenêtre d'animation. Vous avez terminé l'animation pour cette partie de l'exercice.

11. Dans le panneau *Project,* sélectionnez votre ***clip*** d'animation **damage.**



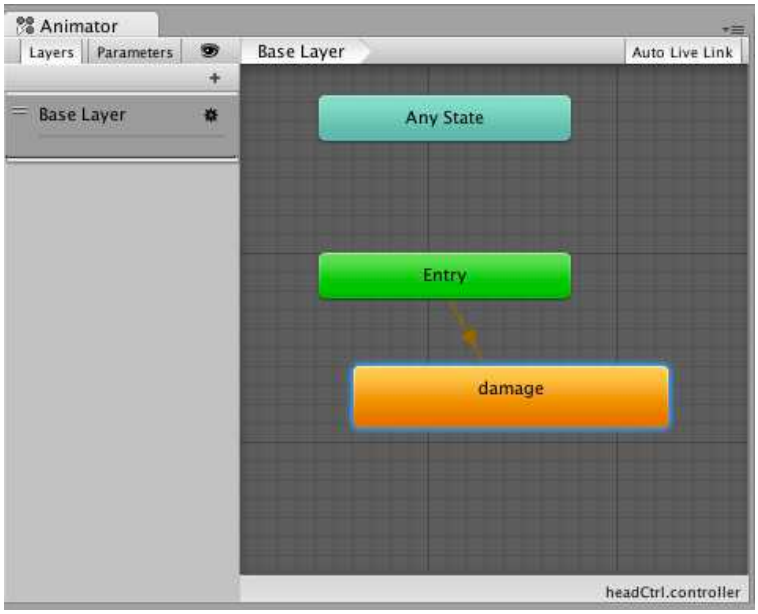
11.1. Dans *l'Inspector*, changez :

11.1.1. Décochez ***Loop Time.***

Nous voulons que ce clip joue seulement une fois et non en boucle.

12. Dans la barre de menus, allez à :

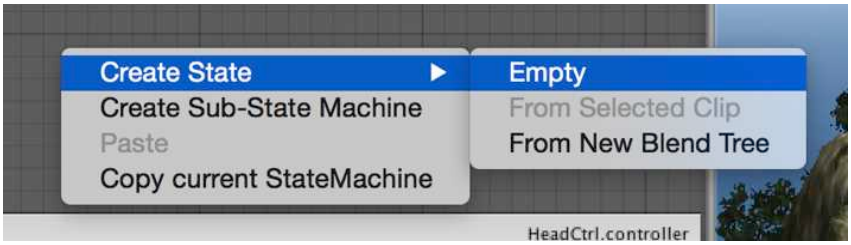
Window | Animator



13. Vous utiliserez un paramètre de type ***Trigger*** afin de déclencher l'animation sur demande.

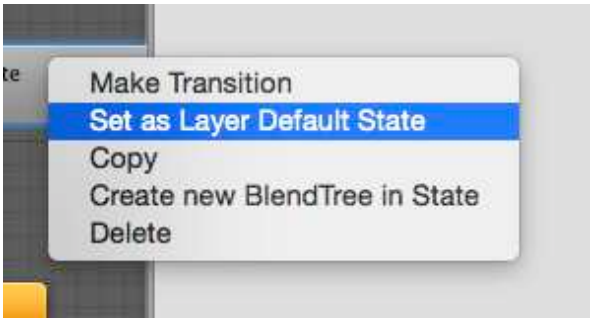
Notez-bien!

Dans cette fenêtre, le champ orangé représente l'animation par défaut. Comme nous ne voulons pas que le tremblement de tête joue continuellement, nous allons créer un nouveau ***clip*** *vide qui sera défini par défaut.*



14. Dans la fenêtre *Animator,* faites un clic secondaire dans un espace vide, puis :

➢ **Create State > Empty**

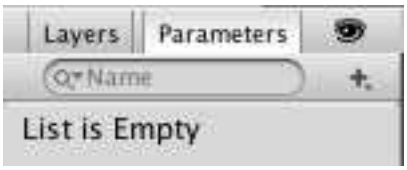


15. Faites un clic secondaire sur le nouveau ***clip*** **New States**, puis appuyez sur :

« Set As Layer Default State ».

Le ***clip*** orange devrait avoir changé de place.

Afin de créer une transition, vous allez ajouter un paramètre (***Parameters***) de type ***Trigger***.



Dans le coin, en haut à gauche de la fenêtre *Animator*, localisez l'onglet nommé **Parameters.**

16. Cliquez sur **Parameters** pour afficher sa liste.

16.1. Appuyez sur le **(+)** et dans la liste qui apparaît, choisissez ***Trigger.***

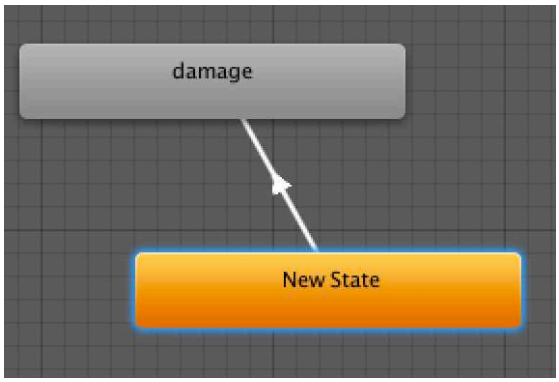
16.1.1. Nommez ce paramètre **damage**.

Vous devez désormais faire une transition entre les clips avec les conditions requises.

17. Faites un clic secondaire sur le ***clip* New State,** puis appuyez sur « Set Transition ».

Une flèche partant du ***clip* New State** apparaîtra.

17.1. Raccordez la flèche à **damage**.



18. Sélectionnez maintenant la flèche entre les deux clips.

18.1. Dans *l'Inspector*, ajoutez une condition en appuyant sur le **(+)**.

18.2. Cette condition doit être :

18.2.1. ***Conditions*= damage**

Nous venons de signaler à Unity® que lorsque le Trigger **damage** est activé, il faut déclencher l'animation **damage**.

Notez-bien!

*Même si le **paramètre** porte le même nom que le **clip damage**, ceux-ci ne sont pas liés ensemble.*

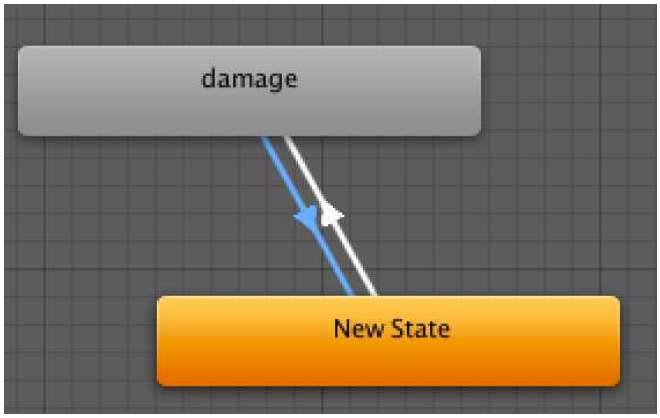
Les deux éléments auraient pu porter des noms différents.

Vous avez désormais un lien pour déclencher l'animation de tremblement, mais vous n'avez aucun lien de retour vers le ***clip*** **New State**.

19. Faites un clic secondaire sur le ***clip*** **damage.**

19.1. Appuyez sur « Set Transition ».

19.2. Raccordez la flèche à **New State.**



20. Fermez cette fenêtre. Vous avez terminé avec le contrôleur d'animation.

Nous sommes désormais prêts à ajouter un nouveau ***script*** pour créer un impact.

21. Faites un clic secondaire dans le panneau *Project,* puis :

- **Create > Javascript**

21.1. Nommez-le **GroundHit.**

22. Placez ce script sur **First Person Controller.**

Ouvrez le script « GroundHit.js ».

Ajoutez le code suivant :

```
var FPMainCamera:Camera;
private var isInPain:boolean = false;
var damageSfx:AudioClip;
var damageImage:UI.Image;
var headAnimator:Animator;
var LavaComponent:UnityStandardAssets.ImageEffects.MotionBlur;

function Update () {
  if (isInPain){
     LavaComponent.blurAmount -= 0.01;
     if (LavaComponent.blurAmount < 0.05)
        unPain();
  }
}

function unPain(){
  isInPain = false;
  LavaComponent.enabled = false;
  LavaComponent.blurAmount = 0.92;
}

function OnCollisionEnter(collision : Collision) {
  if (collision.relativeVelocity.magnitude > 18){
     headAnimator.SetTrigger("damage");
     yield WaitForSeconds(0.05);
     var AudioS = FPMainCamera.GetComponent.<AudioSource>();
     AudioS.clip = damageSfx;
     damageImage.enabled = true;
     AudioS.Play();
     LavaComponent.enabled = true;
     LavaComponent.blurAmount = 0.92;
     isInPain = true;
     Invoke("unDamaged", 0.05);
  }
}

function unDamaged(){
   damageImage.enabled = false;
}
```

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Quand ce script détecte une collision, il vérifie si la magnitude de la vélocité relative est supérieure à 18 (**collision.relativeVelocity.magnitude > 18**). Si oui, il joue un son (**AudioS.Play()**), il affiche un filtre rouge sur l'écran, il active le filtre (***MotionBlur***) sur la caméra. Ensuite, on active (**isInPain**) afin que la fonction (**unPain()**) puisse diminuer l'effet de flou graduellement jusqu'à ce que cette valeur soit plus petite que 0.05. Quand cette valeur est atteinte, on retire les différents effets.

23. Sélectionnez **First Person Controller,** dans votre *Hierarchy.*

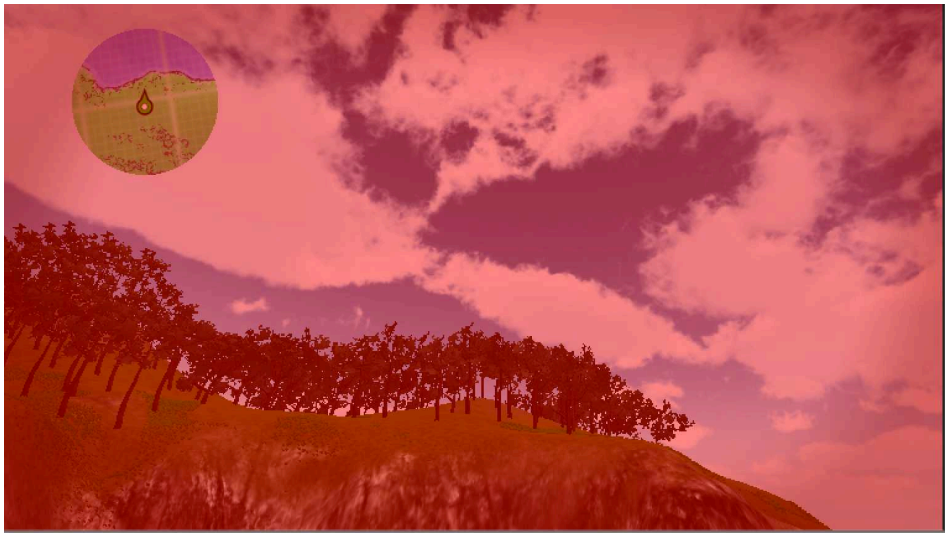
23.1. Dans *l'Inspector*, changez les valeurs :

Ground Hit (Script)
23.1.1. ***FP Main Camera :*** MainCamera
23.1.2. ***Damage Sfx:*** Crac.wav
23.1.3. ***Damage Image***: damageFx
23.1.4. ***Head Animator*** : Head
23.1.5. ***Lava Component*** **:** MainCamera (à nouveau)

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Testez votre jeu.

Sautez de très haut!



...Crac!!!!!

Retournez en mode édition.

Voilà ce qui met fin à cette partie de l'exercice. Vous allez passer à un autre concept très important, c'est à dire un menu de **pause** pour interrompre le déroulement du jeu.

Interfaces utilisateurs (UI) : Ajout d'un menu de pause

Nous avons notre interface pour le gameplay, il faut ajouter un menu de pause comme on le retrouve dans pratiquement tous les jeux. De là, le joueur pourra quitter ou reprendre sa partie.

1. Sélectionnez **CanvasUI** dans le votre *Hierarchy.*

1.1. Allez dans la barre de menus :

GameObject | UI | Text

1.2. Nommez ce texte **Txt_Pause.**

1.3. Dans *l'Inspector,* changez les paramètres suivants :

Rect Transform
1.3.1. ***Pos* : X=** 0**, Y=** 0**, Z=** 0
1.3.2. ***Width*** : 160, ***Height*** : 30

Text (Script)
1.3.3. ***Text*** : PAUSE
1.3.4. ***Font*** : ChrisSILMali-B.ttf
1.3.5. ***Font Style*** : Normal
1.3.6. ***Font Size*** : 140
1.3.7. ***Line Spacing*** : 1
1.3.8. ***Rich Text*** : oui
1.3.9. ***Alignment*** : Centré, milieu

1.3.10. ***Horizontal Overflow*** : Overflow
1.3.11. ***Vertical Overflow*** : Overflow
1.3.12. ***Best Fit*** : Non
1.3.13. ***Color*** : Jaune
1.3.14. ***Material*** : aucun

Vous allez ajouter une ombre portée sur le texte pour le rendre plus visible dans l'interface.

2. Appuyez sur le bouton « Add Component », puis :

UI | Effects | Shadow

2.1. Dans *l'Inspector*, changez **:**

Shadow (Script)

2.1.1. ***Effect Color*** : Noir
2.1.2. ***Effect Distance*** **: X**=8**, Y**=-8
2.1.3. ***Use Graphic Alpha*** **:** Oui

Vous allez ajouter des boutons pour ce menu.

3. Sélectionnez l'image **menu_btn** du *Project* qui se trouve dans le répertoire **Texture.**

3.1. Dans *l'Inspector,* changez :

3.1.1. ***Texture Type*** : Sprite (2D and UI)
3.1.2. ***Sprite Mode*** : Single
3.1.3. ***Pixel Per Unit*** : 100
3.1.4. ***Pivot*** : Top Right
3.1.5. ***Generate Mip Maps*** : Oui
3.1.6. ***Filter Mode*** : Bilinear
3.1.7. ***Max Size*** : 512
3.1.8. ***Format*** : Truecolor

3.2. Appuyez sur « Apply ».

4. Sélectionnez l'image **quit_btn** du *Project* qui se trouve dans le répertoire **Texture.**

4.1. Dans *l'Inspector,* changez :

4.1.1. ***Texture Type*** : Sprite (2D and UI)
4.1.2. ***Sprite Mode*** : Single
4.1.3. ***Pixel Per Unit*** : 100
4.1.4. ***Pivot*** : Bottom Left
4.1.5. ***Generate Mip Maps*** : Oui
4.1.6. ***Filter Mode*** : Bilinear

4.1.7. ***Max Size*** : 512
4.1.8. ***Format*** : Truecolor

4.2. Appuyez sur « Apply ».

Ces deux images ont maintenant des propriétés similaires, à l'exception du ***Pivot.***

À la prochaine étape, on ajoute les boutons à l'interface.

5. Dans *Hierarchy*, sélectionnez **CanvasUI.**

5.1. Dans la barre de menus, allez dans :

GameObject | UI | Button

Un minuscule bouton devrait apparaître quelque part dans l'interface.

5.2. Renommez le bouton **Btn_quit.**

5.3. Dans *l'Inspector,* changez les paramètres suivants :

Image (Script)
5.3.1. ***Source Image*** : quit_btn
5.3.2. ***Color*** : Blanc
5.3.3. ***Material*** : aucun
5.3.4. ***Image Type*** : Simple
5.3.5. ***Preserve Aspect*** : oui
5.3.6. Appuyez sur le bouton « Set Native Size »

Rect Transform
5.3.7. ***Pos*** **: X=** 0**, Y=** 0**, Z=** 0
5.3.8. ***Width*** : 512**, *Height*** : 512
5.3.9. ***Anchors Min*** **: X=** 0, **Y=** 0
5.3.10. ***Anchors Max*** **: X=** 0, **Y** = 0
5.3.11. ***Pivot*** **: X=** 0**, Y=** 0

Button (Script)
5.3.12. ***Interactable*** : oui
5.3.13. ***Transition*** : Color Tint
5.3.14. ***Target Graphic*** : glissez la texture : **Btn_quit** de *Project.*
5.3.15. ***Normal Color*** : Blanc
5.3.16. ***Highlighted Color*** : Jaune
5.3.17. ***Pressed Color*** : Gris Foncé
5.3.18. ***Disabled Color*** : Gris pâle, semi-transparent
5.3.19. ***Color Multiplier*** : 1
5.3.20. ***Fade Duration*** : 0.1

5.3.21. ***Navigation*** : Automatic

6. Sélectionnez l'objet **Text** qui se trouve dans **Btn_quit** du **CanvasUI.**

6.1. Dans l'Inspector, changez :

Rect Transform
6.1.1. ***Pos*** **: X=** 100**, Y=** 0**, Z=** 0
(Il est préférable de conserver ces paramètres pour la fin).

6.1.2. ***Width*** : 92, ***Height*** : 126
6.1.3. ***Anchors Min*** **: X=** 0, **Y=** 0.5
6.1.4. ***Anchors Max*** **: X=** 0, **Y=** 0.5
6.1.5. ***Pivot*** **: X=** 0, **Y=** 0.5

Text (Script)
6.1.6. ***Text*** : QUITTER
6.1.7. ***Font*** : CharisSILMali-B.ttf
6.1.8. ***Font Style*** : Normal
6.1.9. ***Font Size*** : 89
6.1.10. ***Line Spacing*** : 1
6.1.11. ***Rich Text*** : oui
6.1.12. ***Alignment*** : Gauche, milieu

6.1.13. ***Horizontal Overflow*** : Overflow
6.1.14. ***Vertical Overflow*** : Overflow
6.1.15. ***Best Fit*** : Non
6.1.16. ***Color*** : Orange Foncé
6.1.17. ***Material*** : aucun

Vous allez ajouter de l'ombre sur le texte.

6.2. Appuyez sur le bouton « Add Component », puis :

UI | Effects | Shadow

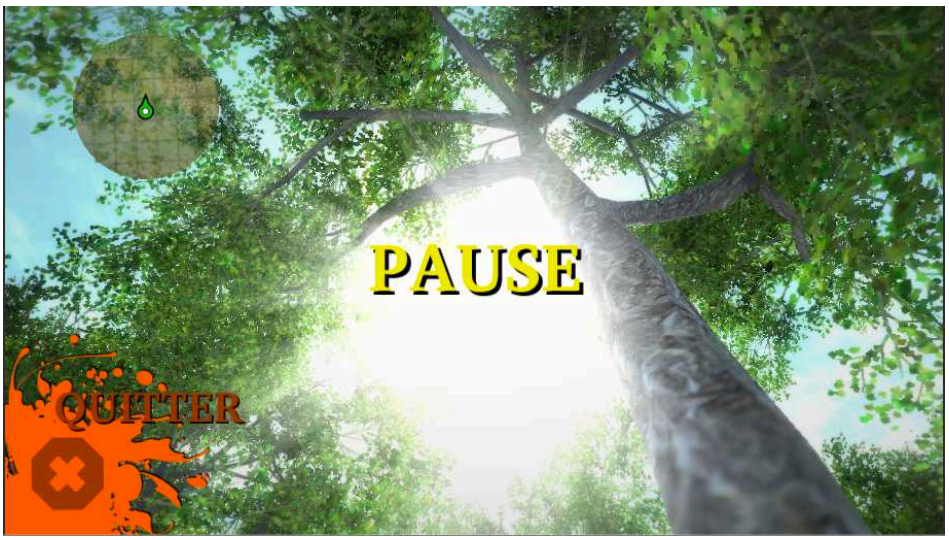
6.3. Dans *l'Inspector*, changez**:**

Shadow (Script)
6.3.1. ***Effect Color*** : Noir, Alpha=255
6.3.2. ***Effect Distance*** **: X**=3**, Y**=-3
6.3.3. ***Use Graphic Alpha*** **:** Oui

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Testez le jeu

Le bouton **quitter** devrait fonctionner en apparence, c'est-à-dire que vous pouvez cliquer dessus et rien ne se produira. Nous y reviendrons ultérieurement.



Retournez en mode édition.

Continuez sur votre lancée avec un deuxième bouton.

7. Dans votre *Hierarchy*, sélectionnez **CanvasUI.**

7.1. Dans la barre de menus, allez à :

GameObject | UI | Button

Un minuscule bouton devrait apparaître quelque part dans l'interface.

7.2. Renommez-le **Btn_resume.**

7.3. Changez les paramètres suivants :

Image (Script)
7.3.1. ***Source Image*** : glissez la texture : **menu_btn** de *Project*
7.3.2. ***Color*** : Blanc
7.3.3. ***Material*** : aucun
7.3.4. ***Image Type*** : Simple
7.3.5. ***Preserve Aspect*** : oui
7.3.6. Appuyez sur le bouton « Set Native Size »

Rect Transform
7.3.7. ***Pos*** **: X=** 0**, Y=** 0**, Z=** 0
(Il est préférable de garder ce paramètre pour la fin).

7.3.8. ***Width*** **:** 512**,** ***Height*** **:** 512
7.3.9. ***Anchors Min*** **: X=** 1**, Y=** 1
7.3.10. ***Anchors Max*** **: X=** 1**, Y=** 1
7.3.11. ***Pivot*** **: X=** 1**, Y=** 1

Button (Script)
7.3.12. ***Interactable*** : oui
7.3.13. ***Transition*** **:** Color Tint
7.3.14. ***Target Graphic*** : Btn_resume
7.3.15. ***Normal Color*** : Blanc
7.3.16. ***Highlighted Color*** : Jaune
7.3.17. ***Pressed Color*** : Gris Foncé
7.3.18. ***Disabled Color*** : Gris pâle, semi-transparent
7.3.19. ***Color Multiplier*** : 1
7.3.20. ***Fade Duration*** : 0.1
7.3.21. ***Navigation*** : Automatic

8. Sélectionnez **Text** du **Btn_resume** de votre *Hierarchy.*

8.1. Dans l'Inspector, changez :

Rect Transform
8.1.1. ***Pos*** **: X=** -100**, Y=** 0**, Z=** 0
(Il est préférable de garder ce paramètre pour la fin).

8.1.2. ***Width*** : 92, ***Height*** : 126
8.1.3. ***Anchors Min*** **: X=** 1, **Y=** 0.5
8.1.4. ***Anchors Max*** **: X=** 1, **Y=** 0.5
8.1.5. ***Pivot*** **: X=** 1, **Y=** 0.5

Text (Script)
8.1.6. ***Text*** : RETOUR
8.1.7. ***Font*** : CharisSILMali-B.ttf
8.1.8. ***Font Style*** : Normal
8.1.9. ***Font Size*** : 89
8.1.10. ***Line Spacing*** : 1
8.1.11. ***Rich Text*** : oui
8.1.12. ***Alignment*** : Droite, milieu
8.1.13. ***Horizontal Overflow*** : Overflow
8.1.14. ***Vertical Overflow*** : Overflow
8.1.15. ***Best Fit*** : Non
8.1.16. ***Color*** : Vert Foncé
8.1.17. ***Material*** : aucun

Vous allez ajouter une ombre portée sur le texte.

8.2. Appuyez sur le bouton « Add Component », puis :

UI | Effects | Shadow

8.3. Changez les paramètres suivants :

Shadow (Script)
8.3.1. ***Effect Color*** : Noir, Alpha= 255
8.3.2. ***Effect Distance*** **: X**= 3**, Y**= -3
8.3.3. ***Use Graphic Alpha*** **:** Oui

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Testez votre jeu.

Les deux boutons devraient être visuellement prêts pour un ***script***.

Retournez en mode édition.

Il est maintenant temps de s'attaquer aux éléments importants et de programmer le comportement du menu.

9. Créez un nouveau ***script*** en positionnant le curseur au-dessus du panneau *Project*. Avec un clic secondaire, faites:

- **Create > Javascript**

9.1. Nommez ce script **pause.**

10. Glissez ce script sur l'objet **EventSystem** qui se trouve dans *Hierarchy*.

Ouvrez le script « pause.js ».

Ajoutez le code suivant :

```
var menuElement:GameObject[];

function Start () {
   initMenu();
}

function initMenu(){
   for (var i=0;i<menuElement.length;i++)
      menuElement[i].SetActive(false);
}
```

Sauvegardez le script et retournez dans l'éditeur.

En détails...

On débute avec une liste de **GameObject** (**menuElement**). L'utilisation de (**[ ]**) dans le code, dicte à Unity® que nous voulons un tableau contenant plusieurs objets. La fonction (**initMenu()**) cache tous les éléments (**SetActive(false)**) de la liste grâce à une boucle (**for**).

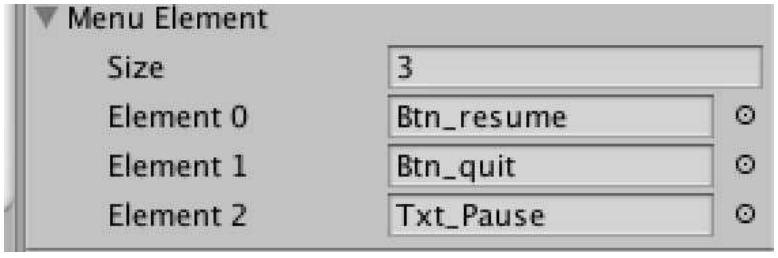
On appelle cette fonction au démarrage du script (**Start()**).

11. Sélectionnez **EventSystem** de *Hierarchy.*

11.1. Dans *l'Inspector*, changez :

11.1.1. ***Menu Element*** : **Size=** 3

• ***Element 0*** : Glissez le bouton **Btn_resume** de *Hierarchy.*
• ***Element 1*** : Glissez le bouton **Btn_quit** de *Hierarchy.*
• ***Element 2*** : Glissez le texte **Txt_Pause** de *Hierarchy.*



Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Testez votre jeu.

Le contenu du menu de **pause** devrait disparaître.

Retournez en mode édition.

Pour faire apparaître le menu, il faudra définir une touche sur laquelle appuyer.

Notez-bien!

*Il existe **deux façons** d'y arriver:*

La méthode simple où l'on inscrit explicitement le nom de la touche sur laquelle appuyer dans le script. Cette approche ne peut être personnalisée par le joueur et se retrouve à être dépendante de la plateforme.

*La seconde méthode consiste à utiliser **Input Manager** qui est utile, beaucoup plus concise et personnalisable.*

12. Dans la barre de menus, allez dans :

Edit | Project settings | Input

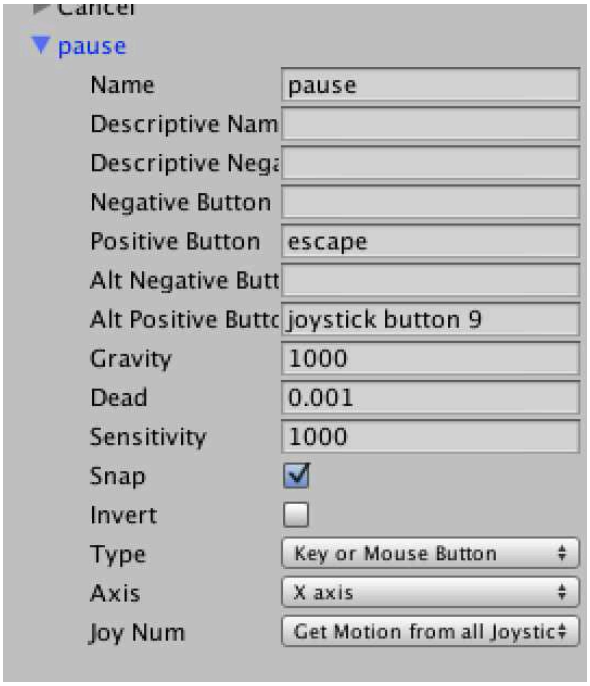
Dans *l'Inspector,* il y a une liste de toutes les **entrées** (***Inputs***) définies par défaut dans Unity®.

Nous allons ajouter une **entrée** à la liste.

13. Ajoutez **1 axe** de plus dans la variable ***Axes* Size.**

À la fin de la liste, un nouvel élément s'est ajouté. Il porte le même nom que celui qui le précède.

13.1. Dans *l'Inspector*, changez les propriétés de cet ***axe*** pour :



13.1.1. ***Name*** **:** pause
13.1.2. ***Positive Button*** **:** escape
13.1.3. ***Alt Positive Button*** : joystick button 9
13.1.4. ***Gravity*** : 1000
13.1.5. ***Dead*** **:** 0.001
13.1.6. ***Sensitivity*** **:** 1000
13.1.7. ***Snap*** **:** oui
13.1.8. ***Invert*** : non
13.1.9. ***Type*** : Key or Mouse Button
13.1.10. ***Joy Num*** : Get motion from all joystick

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Ouvrez le script « pause.js ».

Ajoutez le code suivant :

```
var menuElement:GameObject[];
var InputNameToOpenMenu:String = "pause";
private var menuOpen:boolean = false;

function Start () {
    initMenu();
}

function Update () {
    if (Input.GetButtonUp(InputNameToOpenMenu)){
        switchState();
    }
}

function initMenu(){
    menuOpen = false;
    for (var i=0;i<menuElement.length;i++)
        menuElement[i].SetActive(false);
}

function switchState() {
    menuOpen = !menuOpen;
    for (var i=0;i<menuElement.length;i++)
        menuElement[i].SetActive(menuOpen);
}
```

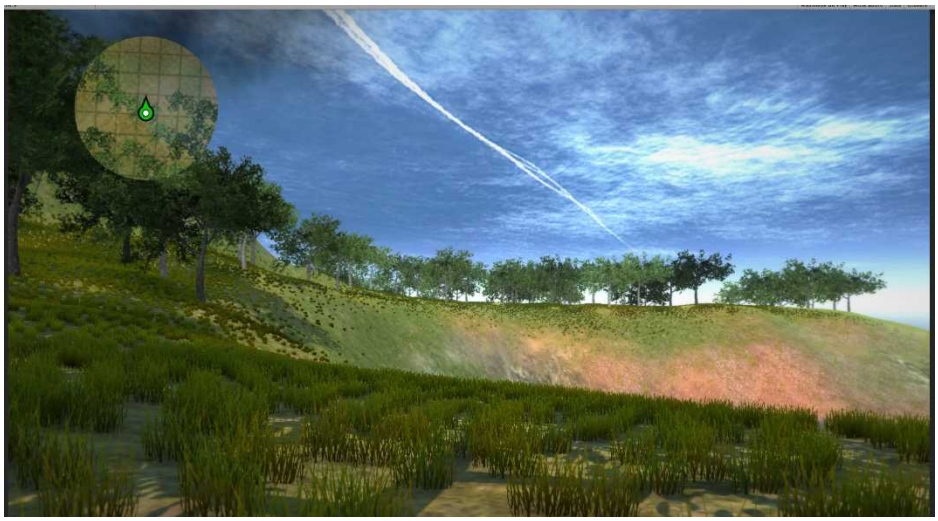
↘ *Les textes en* ***gras*** *sont à ajouter, le reste demeure inchangé.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

La fonction (**Update()**) inspecte si le joueur relâche le bouton (**Input.GetButtonUp**) défini dans la variable (**InputNameToOpenMenu**). On appelle la fonction (**switchState()**), celle-ci inverse la valeur de la nouvelle variable (**menuOpen**), affiche ou cache tous les éléments du menu (**menuElement[i].SetActive**).

Testez votre jeu.



Appuyez sur la touche (**esc** ⎋) du clavier ou le bouton (**Start**) sur une manette de jeu.

Retournez en mode édition.

Le menu devrait s'afficher, mais rien ne s'arrête. Les particules et les vagues sont animées et le joueur peut toujours contrôler la caméra. Alors, nous devons ajouter le code pour arrêter le temps.

Ouvrez le script « pause.js »

Ajoutez le code suivant :

```
var menuElement:GameObject[];
var InputNameToOpenMenu:String = "pause";
private var menuOpen:boolean = false;
var charCtrl:GameObject;
private var oldTimeScale:float = 0.0;

…
function switchState() {
    var RigidCtrl = charCtrl.GetComponent.<UnityStandardAssets.
                                    Characters.FirstPerson.
                                    RigidbodyFirstPersonController>();
    menuOpen = !menuOpen;
    for (var i=0;i<menuElement.length;i++)
        menuElement[i].SetActive(menuOpen);

    var tempoScale = Time.timeScale;
    Time.timeScale = oldTimeScale;
    oldTimeScale = tempoScale;
    RigidCtrl.enabled = !menuOpen;
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

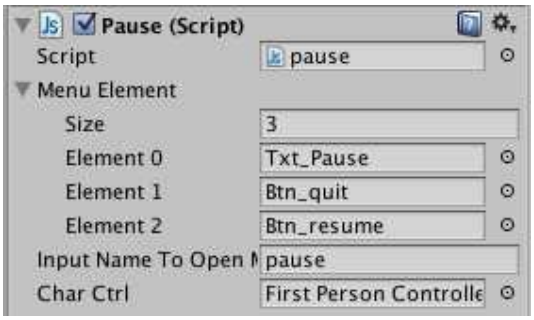
Sauvegardez le script et retournez dans l'éditeur.

En détails…

Ce nouveau code permute la vitesse du jeu entre 0.0 et la vitesse originale (**Time.timeScale**). Il gèle le component (**FirstPerson.RigidbodyFirstPersonController**) qui contrôle le déplacement du joueur.

14. Ensuite, sélectionnez **EventSystem** dans *hierarchy.*

14.1. Ajoutez l'élément suivant dans la variable :



Pause (Script)
14.1.1. ***Char Ctrl*** **:** glissez **First Person Controller** de *Hierarchy.*

Sauvez votre scène (**CTRL+S**) ou (⌘**+S**) sur Mac.

Testez votre jeu.

Le comportement du menu de **pause** est maintenant fonctionnel, mais certains éléments dont la carte, devraient être cachés durant cette **pause**.

Retournez en mode édition.

Ouvrez le script « pause.js ».

Ajoutez le code suivant :

```
var menuElement:GameObject[];
var hideWhenOpen:GameObject[];
var InputNameToOpenMenu:String = "pause";
private var menuOpen:boolean = false;
var charCtrl:GameObject;
…

function switchState() {
    var RigidCtrl = charCtrl.GetComponent.<UnityStandardAssets.
                                    Characters.FirstPerson.
                                    RigidbodyFirstPersonController>();
    menuOpen = !menuOpen;
    for (var i=0;i<menuElement.length;i++)
        menuElement[i].SetActive(menuOpen);
    for (var ii=0;ii<hideWhenOpen.length;ii++)
        hideWhenOpen[ii].SetActive(!menuOpen);
    var tempoScale = Time.timeScale;
    Time.timeScale = oldTimeScale;
    oldTimeScale = tempoScale;
    RigidCtrl.enabled = !menuOpen;
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails…

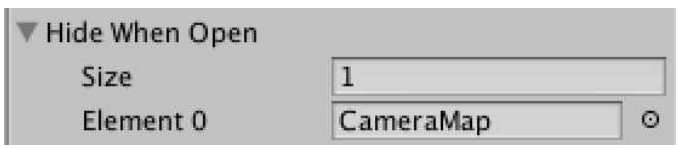
Quand le menu est ouvert, la nouvelle boucle (**for**) va passer dans la liste de tous les objets pour les cacher. Elle les affichera, lorsque le menu sera fermé.

15. Sélectionnez l'objet **EventSystem** de *Hierarchy.*

15.1. Dans *l'Inspector*, changez :

15.1.1. ***Hide When Open*** **: Size=** 1

• ***Element 0*** **:** Glissez la caméra **CameraMap** de *Hierarchy.*



Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Testez votre jeu.

Si le menu est actif, la carte devrait disparaître.

Retournez en mode édition.

Vous allez ajouter un son pour l'ouverture du menu.

Ouvrez le script « pause.js ».

Ajoutez le code suivant :

```
var menuElement:GameObject[];
var hideWhenOpen:GameObject[];
var InputNameToOpenMenu:String = "pause";
private var menuOpen:boolean = false;
var charCtrl:GameObject;
var sfxPause:AudioClip;
var FPMainCamera:Camera;
…

function switchState() {
    var RigidCtrl = charCtrl.GetComponent.<UnityStandardAssets.
                                    Characters.FirstPerson.
                                    RigidbodyFirstPersonController>();
    var AudioS = FPMainCamera.GetComponent.<AudioSource>();
    menuOpen = !menuOpen;
    for (var i=0;i<menuElement.length;i++)
        menuElement[i].SetActive(menuOpen);
    for (var ii=0;ii<hideWhenOpen.length;ii++)
        hideWhenOpen[ii].SetActive(!menuOpen);
    AudioS.PlayOneShot(sfxPause);
    var tempoScale = Time.timeScale;
    Time.timeScale = oldTimeScale;
    oldTimeScale = tempoScale;
    RigidCtrl.enabled = !menuOpen;
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails…

Un son joue lorsque l'état du menu change (**PlayOneShot(sfxPause)**).

16. Sélectionnez l'objet **EventSystem** de *Hierarchy*.

16.1. Dans l'Inspector, ajoutez les éléments suivants :

Pause (Script)
16.1.1. ***Sfx Pause*** : glissez le son **Pause.wav** de *Project.*
16.1.2. ***FP Main Camera*** : glissez **Main Camera** de *Hierarchy.*

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Testez votre jeu.

Le son devrait jouer à l'ouverture ou à la fermeture du menu.

Retournez en mode édition.

Ouvrez le script « pause.js ».

Ajoutez la fonction suivante :

```
…
function Quit(){
 Application.Quit();
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

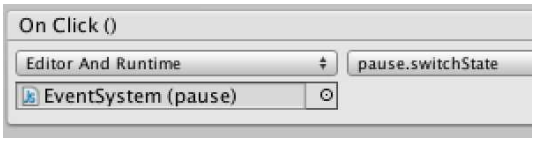
17. Sélectionnez le bouton **Btn_resume** dans *Hierarchy.*

17.1. Dans *l'Inspector*, localisez la section **On Click()**.

17.2. Ajoutez un nouvel évènement en appuyant sur **(+).**

17.2.1. **Dans la case du bas** : Glissez **EventSystem** de *Hierarchy.*
17.2.2. **Dans la case de droite :** Sélectionnez la fonction :

Pause | switchState()



En détails…

Nous venons de trouver le bouton qui indique où se trouve le script (**EventSystem**) et quelle fonction appeler quand on clique sur (**switchState()**).

18. Sélectionnez le bouton **Btn_quit** dans *Hierarchy.*

18.1. Puis dans *l'Inspector,* localisez la section **On Click()**.

18.2. Ajoutez un nouvel évènement en appuyant sur **(+).**

18.2.1. **Dans la case du bas** : Glissez **EventSystem** de *Hierarchy.*
18.2.2. **Dans la case de droite :** Sélectionnez la fonction :

Pause | Quit ()

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Testez votre jeu.

Le bouton « retour » fonctionne. Le bouton « quitter » fonctionne uniquement dans une version distribuable.

Retournez en mode édition.

Vous venez de compléter cet exercice.

Conseil!

Tentez de créer un menu principal comme vous l'avez fait dans le chapitre 7. Essayez d'exporter votre jeu en version redistribuable, comme vous l'avez fait dans le chapitre 5.

Codes complétés

Voici les codes complets non abrégés de l'exercice précédent:

GroundHit

```
#pragma strict

var FPMainCamera:Camera;
private var isInPain:boolean = false;
var damageSfx:AudioClip;
var damageImage:UI.Image;
var headAnimator:Animator;
var LavaComponent:UnityStandardAssets.ImageEffects.MotionBlur;

function Update () {
  if (isInPain){
     LavaComponent.blurAmount -= 0.01;
     if (LavaComponent.blurAmount < 0.05)
        unPain();
  }
}

function unPain(){
   isInPain = false;
   LavaComponent.enabled = false;
   LavaComponent.blurAmount = 0.92;
}

function OnCollisionEnter(collision : Collision) {
   if (collision.relativeVelocity.magnitude > 18){
      headAnimator.SetTrigger("damage");
      yield WaitForSeconds(0.05);
      var AudioS = FPMainCamera.GetComponent.<AudioSource>();
      AudioS.clip = damageSfx;
      damageImage.enabled = true;
      AudioS.Play();
      LavaComponent.enabled = true;
      LavaComponent.blurAmount = 0.92;
      isInPain = true;
      Invoke("unDamaged", 0.05);
   }
}

function unDamaged(){
   damageImage.enabled = false;
}
```

pause

```
#pragma strict

var menuElement:GameObject[];
var hideWhenOpen:GameObject[];
var InputNameToOpenMenu:String = "pause";
private var menuOpen:boolean = false;
var charCtrl:GameObject;
var sfxPause:AudioClip;
var FPMainCamera:Camera;

private var oldTimeScale:float = 0.0;

function Start () {
    initMenu();
}

function Update () {
  if (Input.GetButtonUp(InputNameToOpenMenu)){
     switchState();
  }
}

function initMenu(){
  menuOpen = false;
  for (var i=0;i<menuElement.length;i++)
     menuElement[i].SetActive(false);
}

function switchState() {
  var RigidCtrl = charCtrl.GetComponent.<UnityStandardAssets.
                                  Characters.FirstPerson.
                                  RigidbodyFirstPersonController>();
  var AudioS = FPMainCamera.GetComponent.<AudioSource>();
  menuOpen = !menuOpen;
  for (var i=0;i<menuElement.length;i++)
     menuElement[i].SetActive(menuOpen);
  for (var ii=0;ii<hideWhenOpen.length;ii++)
     hideWhenOpen[ii].SetActive(!menuOpen);
  AudioS.PlayOneShot(sfxPause);
  var tempoScale = Time.timeScale;
  Time.timeScale = oldTimeScale;
  oldTimeScale = tempoScale;
  RigidCtrl.enabled = !menuOpen;
}

function Quit(){
  Application.Quit();
}
```

lava

```
#pragma strict

var lavaEffect:UI.Image;
var screamInPain:AudioClip;
var FPMainCamera:Camera;
var LavaComponent:UnityStandardAssets.ImageEffects.MotionBlur;

function OnTriggerEnter(col:Collider){
  var audioS = FPMainCamera.GetComponent.<AudioSource>();
  if (col.gameObject.tag == "Player"){
     audioS.clip = screamInPain;
     LavaComponent.enabled = true;
     audioS.Play();
     lavaEffect.enabled = true;
     yield WaitForSeconds (audioS.clip.length);
     Application.LoadLevel(Application.loadedLevel);
  }
}
```

underwater

```
#pragma strict

var underwaterEffect:UI.Image;
var waterObj:Transform;
var diveSound:AudioClip;
var ObjectToHide:GameObject[];

private var isUnder:boolean = false;

function Update () {
  var audioS = GetComponent.<AudioSource>();
  if (transform.position.y < waterObj.position.y && ! isUnder){
     underwaterEffect.enabled = true;
     audioS.clip = diveSound;
     audioS.Play();
     isUnder = true;
     for each (GameObject in ObjectToHide)
        GameObject.active = false;
  }
  if (transform.position.y > waterObj.position.y && isUnder){
     underwaterEffect.enabled = false;
     isUnder = false;
     for each (GameObject in ObjectToHide)
        GameObject.active = true;
  }
}
```

Chapitre 10 - Jeu #3: Incendie

Matière couverte dans ce chapitre

- **Assemblage de la scène**
- **Intégration des animations importées d'un FBX**
- **Assemblage des animations dans *Mecanim***
- **Création d'émetteurs de particules**
- **Ajouts des éléments du jeu**

Avant de débuter

FICHIERS REQUIS

Vous aurez besoin du paquet : **Chapitre10-11.unitypackage** qui contient tous les éléments graphiques de l'exercice.

Atelier pratique

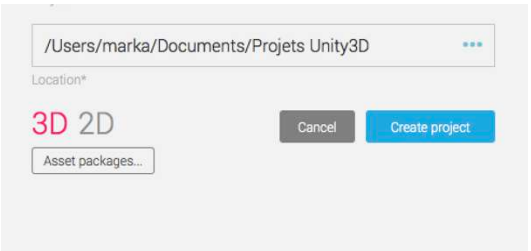
Les éléments de la scène

1. Ouvrez Unity®.

2. Cliquez sur le bouton « New Project ».

3. Sauvez votre projet sur un disque dur local en inscrivant le nom **Incendie.**

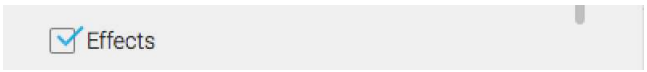
3.1. Ce sera un projet **3D.**

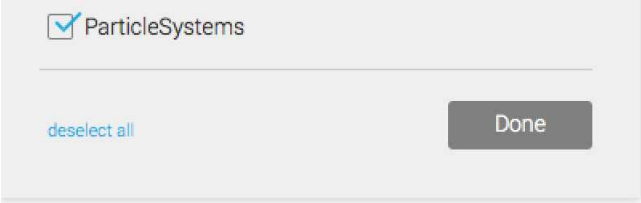


3.2. Cliquez sur « Asset Packages… ».

3.3. Sélectionnez les paquets :

- **Effects**
- **ParticleSystems**





4.Appuyez sur « Done » pour retourner à l'écran de démarrage.

5.Appuyez sur « Create Project » pour débuter.

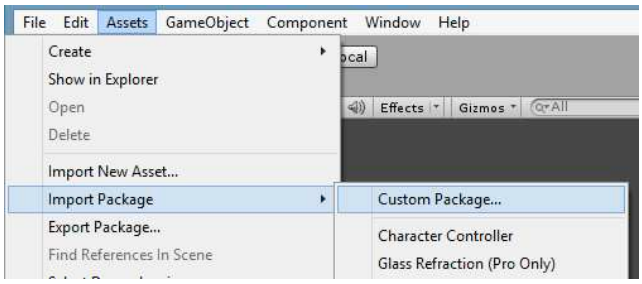
6. Lorsque le projet est ouvert, **sauvegardez votre scène** (**CTRL+S**) ou (**⌘+S**) sur Mac.

6.1. Nommez-la **Game.**

Lorsque la scène est chargée, il faut importer les ***Assets*** de l'exercice.

7. Allez dans la barre de menus :

Assets | Import package | Custom Package…



7.1. Sélectionnez **Chapitre10-11.unitypackage**, importez son contenu.

7.1.1. Appuyez sur « Import ».

Vous aurez accès ces répertoires ainsi que leur contenu.

Fonts
- **CharisSILMali-B.ttf**

Materials
- **Face3D**
- **Hairs_tex**
- **Jacket_tex**
- **Lava**
- **Pants_tex**
- **Stars**
- **Texture_Atlas**
- **Vase**
- **Vase Break**

Meshes
- **Building.FBX**
- **perso.FBX**
- **Vase.FBX**

Prefab
- **Vase_prefab**

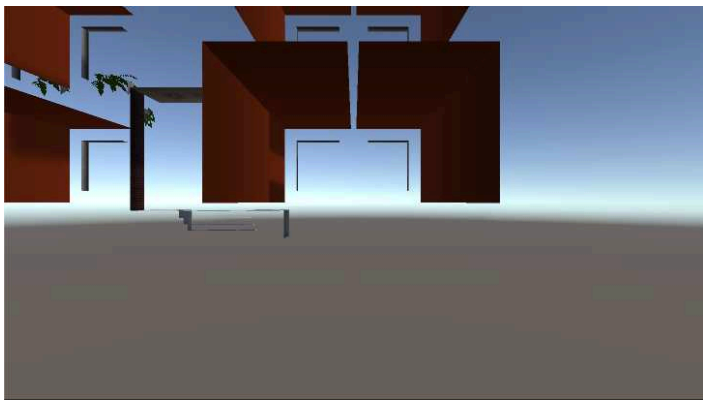
Scripts
- **Character.js**
- **Game.js**
- **Junk.js**
- **Vase.js**

Standard Assets
- **(Le contenu importé)**

Textures
- **Eclats.tga**
- **Face3D.tga**
- **Lava.tga**
- **Star.tga**
- **Texture_Atlas.tga**
- **Vase.tga**

8. À partir de *Project,* glissez l'objet **building** du répertoire **Meshes** vers *Hierarchy*.

Présentement, le building est à l'envers dans la scène.



9. Sélectionnez le **building** dans la fenêtre *Hierarchy.*

9.1. Dans *l'Inspector,* changez :

Transform
9.1.1. ***Rotation*** **: X=** 0, **Y=** 180, **Z=** 0

Vous allez maintenant ajouter un personnage dans la scène.

Ajout du personnage

10. À partir de *Project*, glissez l'objet **perso** du répertoire **Meshes** vers *Hierarchy*.

Changez la ***Position*** et la ***Rotation*** du personnage pour qu'il regarde en direction de la caméra.

11. Sélectionnez l'objet **perso** dans la fenêtre *Hierarchy.*

11.1. Dans *l'Inspector,* changez les valeurs :

Transform
11.1.1. ***Position*** **: X=** 3.85, **Y=** 0, **Z=** -0.6
11.1.2. ***Rotation*** **: X=** 270, **Y=** 180, **Z=** 0

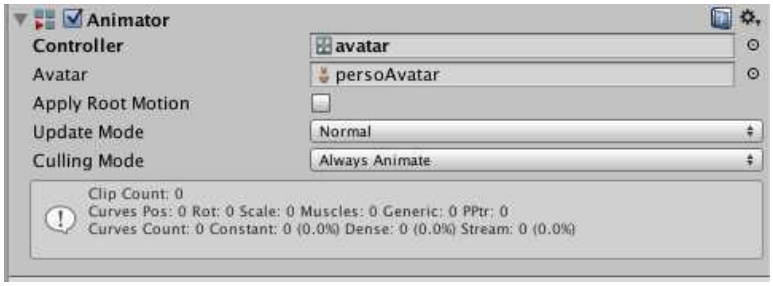
Vous devez maintenant ajouter un ***Animator Controller*** pour assembler les différentes animations du **perso.**

12. Faites un clic secondaire dans *Project,* puis :

➢ **Create > Animator Controller**

12.1. Nommez-le **avatar.**

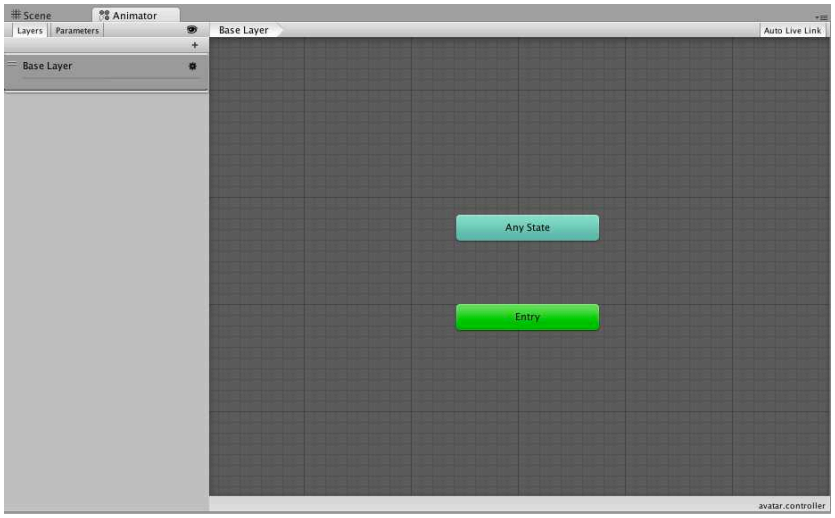
12.2. Glissez le ***Controller*** **avatar** sur l'objet **perso** dans *Hierarchy*.



Ouvrez maintenant la fenêtre *Animator.*

13. Dans la barre de menus, allez dans :

Window | Animator



Pour ce jeu, vous devez utiliser deux paramètres :

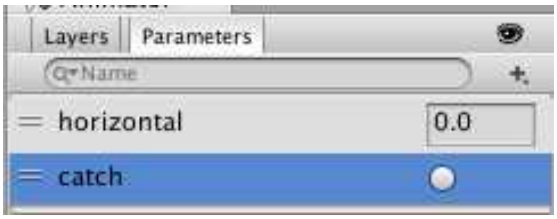
- **Le premier pour le déplacement latéral du personnage**
- **Le second durant l'attrapée du vase**

14. Dans la fenêtre *Animator*, localisez l'onglet ***Parameters*** qui se trouve dans le coin, en haut à gauche.

14.1. Appuyez sur **(+)** pour ajouter un paramètre de type ***Float.***

14.1.1. Nommez ce paramètre **horizontal** en lettres minuscules.
c'est important

15. Appuyez sur **(+)** pour créer un second paramètre de type ***Trigger.***



15.1. Nommez-le **catch** en lettres minuscules.

Vous allez configurer *l'**Animator Controller*** comme ceci :

- **L'intro** jouera par défaut au lancement de l'application.
- Elle fera une transition vers un ***blend tree.***
- Le ***blend tree*** contiendra 3 animations pour le déplacement dans les différentes directions.
- Un second ***Layer*** jouera l'animation de l'attrapée, indépendamment des déplacements.

16. Dans *Project,* localisez le modèle **perso** dans le répertoire **Meshes.**

16.1. Dans *Project*, affichez les éléments de ce modèle en appuyant sur la flèche à côté de l'icône.

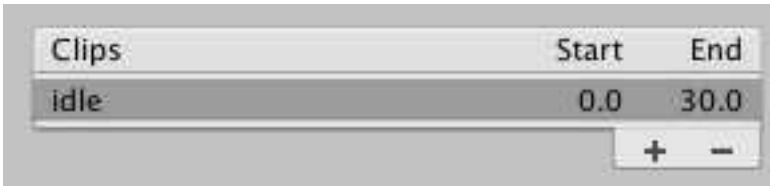
Ce modèle contient une animation nommée **idle**, qui est découpée et ajustée sur un ***rig generic***.

Vous devez ajouter 4 animations additionnelles :

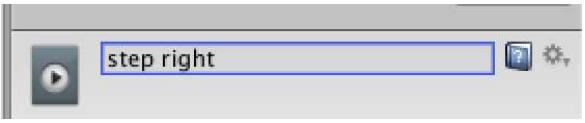
- **catch**
- **intro**
- **step left**
- **step right**

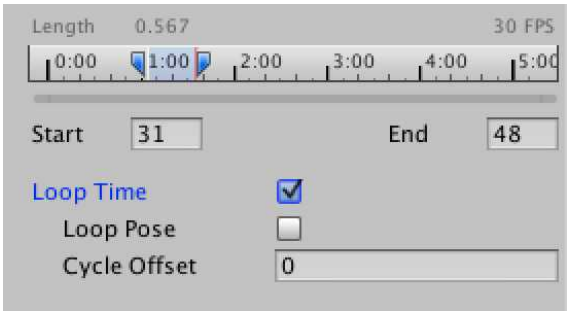
16.2. Dans *l'Inspector,* cliquez sur le bouton « Animations ».

17. Dans la section ***Clips,*** appuyez sur le bouton **(+)** pour créer un nouveau ***clip d'animation***.



17.1. Nommez-le **step right.**





17.2. Dans *l'Inspector*, changez les paramètres suivants :

17.2.1. ***Start*** : 31.
17.2.2. ***End*** : 48.
17.2.3. ***Loop Time*** : oui.

17.3. Appuyez sur le bouton « Apply ».

18. Appuyez sur le bouton **(+)** pour créer un nouveau clip d'animation.

18.1. Nommez-le **step left.**

18.2. Dans *l'Inspector*, changez les paramètres suivants :

18.2.1. ***Start*** : 49.
18.2.2. ***End*** : 66.
18.2.3. ***Loop Time*** : oui.

18.3. Appuyez sur le bouton « Apply ».

19. Appuyez sur le bouton **(+)** pour créer un nouveau ***clip d'animation***.

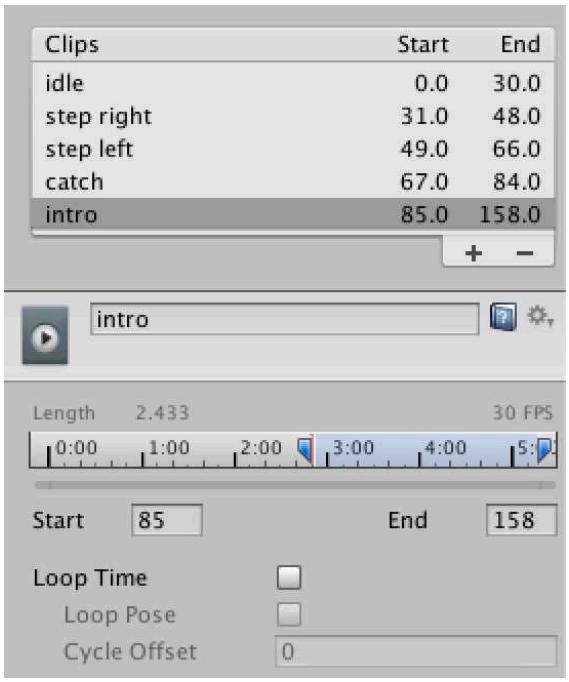
19.1. Nommez-le **catch.**

19.2. Dans *l'Inspector*, changez les paramètres suivants :

19.2.1. ***Start*** : 67.
19.2.2. ***End*** : 84.
19.2.3. ***Loop Time*** : non.

19.3. Appuyez sur le bouton « Apply ».

20. Appuyez sur le bouton **(+)** pour créer un nouveau clip d'animation.



20.1. Nommez-le **intro.**

20.2. Dans *l'Inspector*, changez les paramètres suivants :

20.2.1. ***Start*** : 85.
20.2.2. ***End*** : 158.
20.2.3. ***Loop Time*** : non.

20.3. Appuyez sur le bouton « Apply ».

21. Dans le panneau *Hierarchy,* sélectionnez **perso.**

22. Dans la barre de menus, allez dans :

Window | Animator

23. À partir du panneau *Project,* glissez le ***clip*** **intro** dans la fenêtre *Animator.*

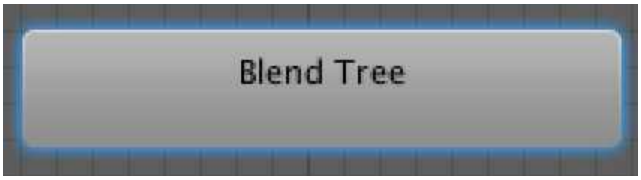
Notez-bien!

C'est l'animation de départ, il est donc important que celle-ci soit orangée.

Vous ajoutez un ***Blend Tree*** pour le déplacement de l'avatar.

24. Faites un clic secondaire dans la fenêtre *Animator,* puis :

- **Create State > From New Blend Tree**



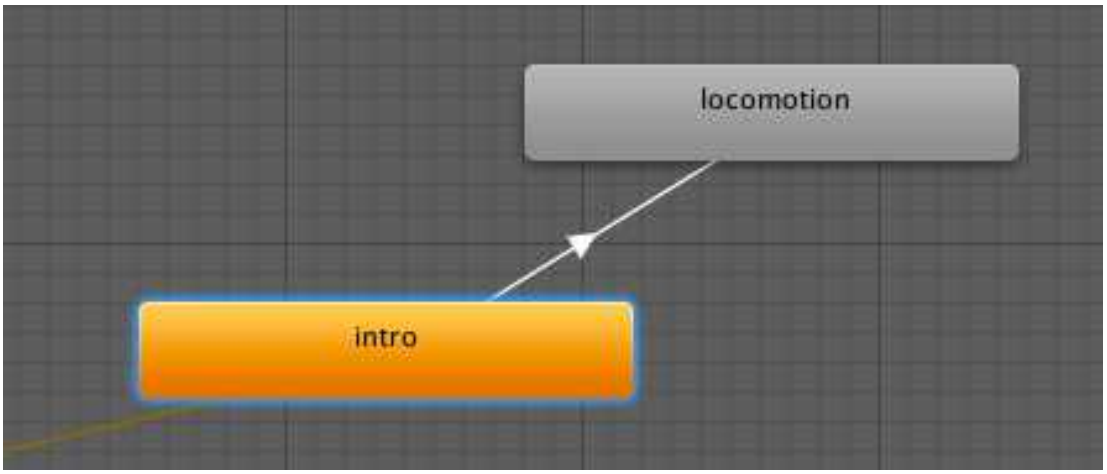
24.1. Nommez-le **locomotion** en lettres minuscules. *Important!*

25. Faites un clic secondaire sur le clip **intro.**

25.1. Appuyez sur « Make Transition ».

Une flèche sera créée à partir du ***clip*** **intro.**

25.2. Raccordez **intro** à **locomotion.**



Notez-bien!

Par défaut, il faut laisser les paramètres de transition en place, car vous devrez faire une transition lorsque l'animation ***intro*** sera *terminée.*

26. Faites un double clic sur **locomotion** pour afficher ces détails.

27. Sélectionnez le ***clip :*** **Blend Tree** dans la fenêtre *Animator.*

27.1. Dans *l'Inspector,* appuyez sur **(+)** dans la section **Motion** puis « Add Motion Field ».

28. Refaites cette étape 2 fois pour obtenir 3 champs ***motion fields.***

28.1. Décochez ***Automate Thresholds.***

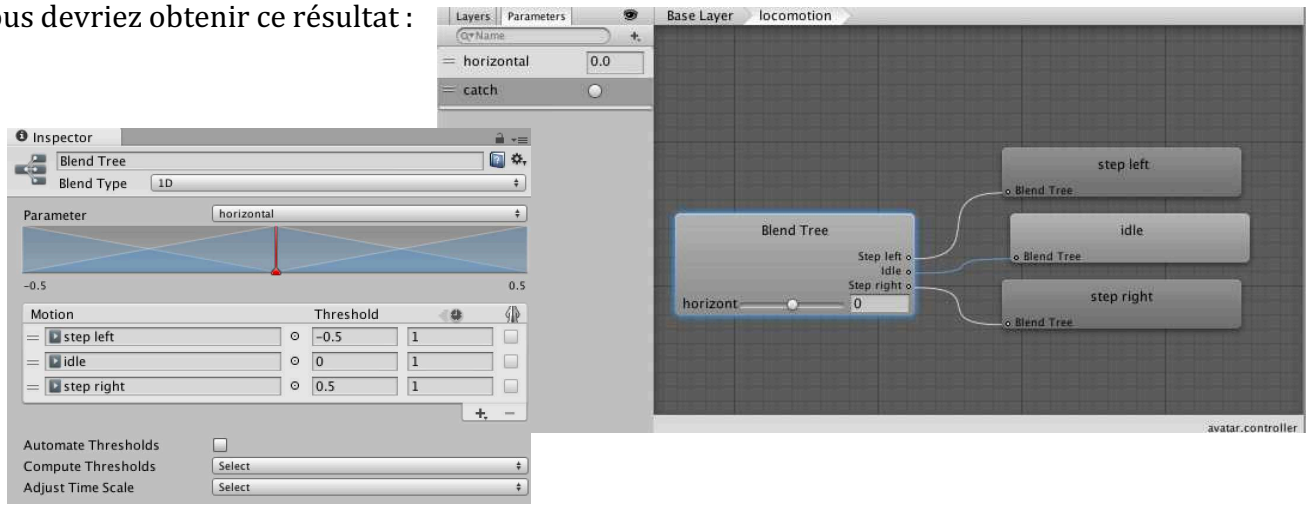
29. Glissez les ***clips*** suivants qui sont dans l'objet **perso** de *Project.*

29.1. *c*hangez les valeurs de ***Thresholds*** :

Motion
29.1.1. **step left, *Threshold* :** -0.5
29.1.2. **idle, *Threshold* :** 0
29.1.3. **step right, *Threshold* :** 0.5

30. Changez la valeur de ***Parameter*** pour **horizontal.**

Vous devriez obtenir ce résultat :



Vous avez tout ce qui est nécessaire pour utiliser les animations de déplacement.

Comme l'attrapée d'un vase est indépendante des actions de déplacement, vous allez ajouter un second ***layer*** qui sera indépendant du premier.

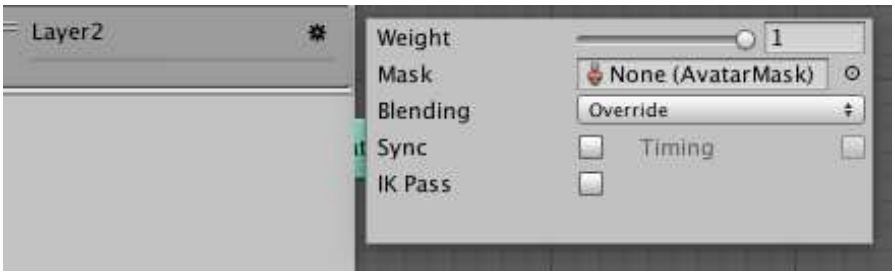
31. Dans le coin en haut à gauche de la fenêtre *Animator*, appuyez sur l'onglet intitulé « Layers ».

31.1. Appuyez sur **(+)** pour ajouter une 2e couche d'animations.

31.2. Nommez-la **Layer2.**

31.3. Dans *l'Inspector*, changez :

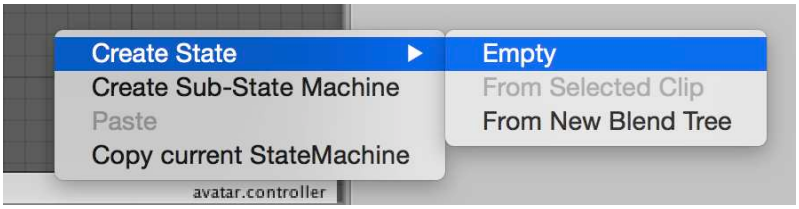
31.3.1. ***Weight*** : 1
31.3.2. ***Mask*** : Aucun
31.3.3. ***Blending*** : Override.



Sur ce nouveau ***layer***, vous allez d'abord ajouter un ***clip d'animation*** vide.

32. Faites un clic secondaire dans la fenêtre *Animator.*

➢ **Create State > Empty**



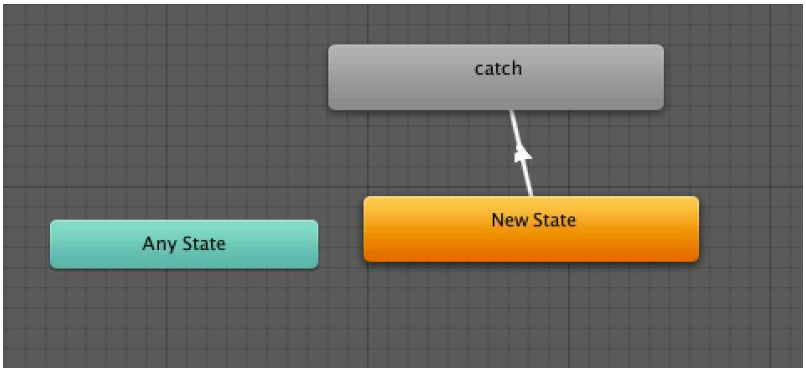
Le nom de ce ***clip*** n'a pas d'importance, car il restera vide.

33. Glissez *l'**animation*** **catch** du modèle **perso** de *Project,* dans la fenêtre *Animator.*

34. Faites un clic secondaire sur le ***clip*** **New State** puis « Make Transition ».

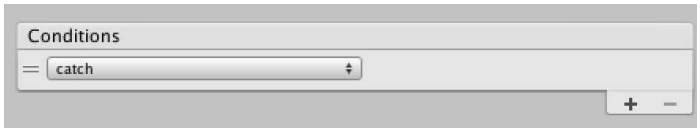
34.1. Raccordez le ***clip*** **New State** à **catch.**

35. Sélectionnez la **transition :**

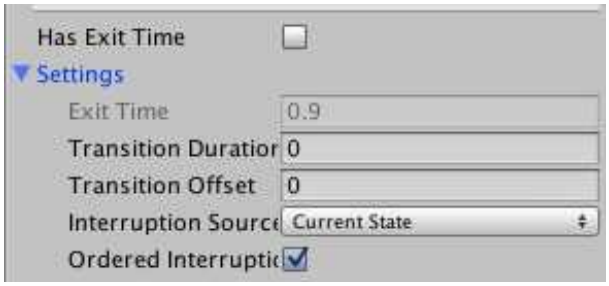


35.1. Dans *l'Inspector,* ajoutez une nouvelle condition en appuyant sur le **(+).**

35.2. Changez la condition pour :



35.2.1. ***Conditions*** = **catch**
35.2.2. ***Has Exit Time*** = non



Settings
35.2.3. ***Transition Duration*** : 0
35.2.4. **Transition Offset** : 0
35.2.5. **Interruption Source** : Current State

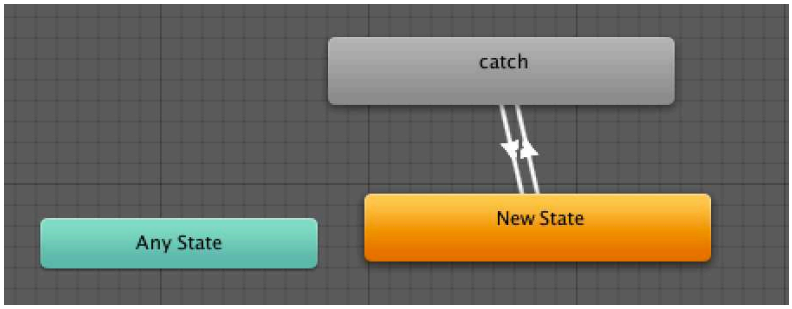
Notez-bien!

*Ce **clip** jouera uniquement quand le paramètre **catch** sera **activé** et sans transition.*

36. Faites un clic secondaire sur le ***clip*** **catch** puis **Make Transition.**

36.1. Raccordez **catch** à **New State**.

36.2. Laissez les ***conditions*** par défaut.



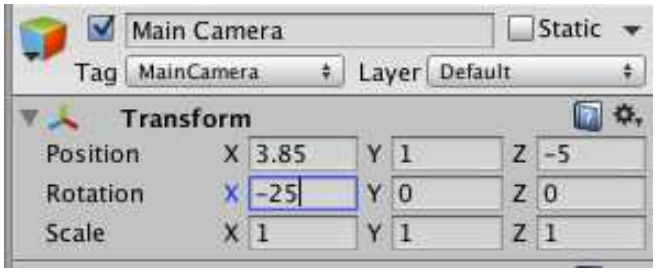
Maintenant que vos ***animations*** sont prêtes, vous allez vous attaquer à l'ambiance de la scène.

Vous allez tout d'abord replacer la position **Main Camera**.

37. Sélectionnez **Main Camera** dans *Hierarchy.*

37.1. Dans *l'Inspector,* changez :

Transform
37.1.1. ***Position*** **: X**= 3.85, **Y**= 1**, Z**= -5
37.1.2. ***Rotation*** **: X**= -25



Vous constatez que le plan de caméra est bon, mais que le ciel est trop bleu. Ajoutez alors une ambiance pour le jeu.

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Le jeu nécessite 4 systèmes de particules :

- **Une fumée dense qui sort du 3e étage.**
- **Des étoiles quand on attrape un vase.**
- **Un large nuage de fumée quand un débris touche le joueur ou le sol.**
- **Des éclats quand un vase touche le sol.**

C'est important de commencer par la fumée du building, car elle lance le jeu. Sans elle, il serait difficile de comprendre le thème du jeu.

Les effets spéciaux

38. Dans la barre de menus, allez dans :

GameObject | Particles System

Par défaut, les particules sont génériques. Il faut modifier ses paramètres pour obtenir l'effet voulu.

39. Sélectionnez l'objet **Particles System.**

39.1. Nommez l'émetteur **Dense_smoke.**

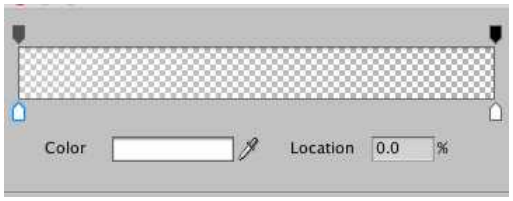
39.2. Dans *l'Inspector,* changez :

Transform
39.2.1. ***Position* : X**= 3.85, **Y**= 5.3, **Z**= 1.59
39.2.2. ***Rotation* : X**= 302, **Y**= 180, **Z**= 180

Paramètres de base
39.2.3. ***Looping*** : oui
39.2.4. ***Prewarm*** : non
39.2.5. ***Start Lifetime*** : 2.83
39.2.6. ***Start Size* :** 10.94
39.2.7. ***Start Rotation* :** 37.75
39.2.8. ***Start Color* :** Blanc
39.2.9. ***Gravity Multiplier* :** 0.28
39.2.10. ***Simulation Space* :** World

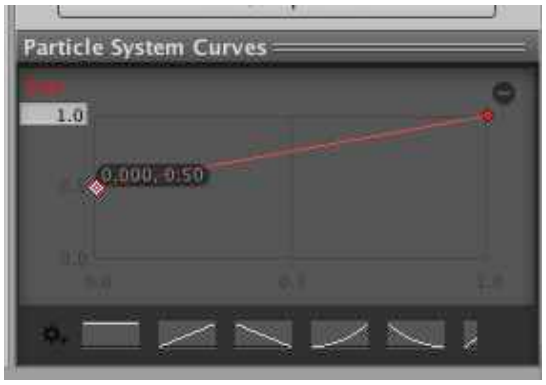
Emission
39.2.11. ***Rate* :** 40

Shape
39.2.12. ***Shape* :** Box
39.2.13. ***Box* : X=** 11, **Y=** 1, **Z=** 1



Color over Lifetime
39.2.14.***Couleur de début*** : Blanc, ***alpha***= 80
39.2.15.***Couleur de fin*** : Blanc, ***alpha***= 0

Size over Lifetime
39.2.16. **Ligne ascendante** de 0.5 à 1.0



Renderer
39.2.17. ***Material* :** Recherchez **ParticleSmokeBlack.**
39.2.18. ***Cast Shadows* :** non
39.2.19. ***Receive Shadows* :** non (c'est plus économique comme ça).

Laissez les valeurs par défaut pour les autres variables.

Vous allez maintenant créer un effet qui se déclenchera quand le joueur touchera un vase.

40. Dans la barre de menus, allez dans :

GameObject | Particles System

41. Sélectionnez l'objet **Particles System.**

41.1. Nommez l'émetteur **Particle_catch.**

41.2. Dans *l'Inspector,* changez :

Paramètres de base
41.2.1. ***Duration*** : 0.10
41.2.2. ***Looping*** : non
41.2.3. ***Start Lifetime*** : 0.3
41.2.4. ***Start Speed*** : 5
41.2.5. ***Start Size*** : 0.5
41.2.6. ***Start Rotation*** : 25
41.2.7. ***Simulation Space*** : Local
41.2.8. ***Play On Awake*** : non

Emission
41.2.9. ***Rate*** : 400

Shape
41.2.10. ***Shape*** : Sphere
41.2.11. ***Radius*** : 0.27
41.2.12. ***Emit from Shell*** : Yes

41.2.13. ***Random Direction*** : Yes

Color over Lifetime
41.2.14. ***Couleur de début*** *:* Blanc, ***alpha***= 255
41.2.15. ***Couleur à 0.85 (%)*** *:* Blanc, ***alpha***= 255
41.2.16. ***Couleur de fin*** : Blanc, ***alpha***= 0

Renderer
41.2.17. ***Material*** **:** Stars
41.2.18. ***Cast Shadows*** : non
41.2.19. ***Receive Shadows*** : non

Vous devriez obtenir ce résultat :

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Vous allez créer un système de particules pour activer la collision avec des boules de feu.

42. Dans la barre de menus, allez dans :

GameObject | Particles System

42.1. Nommez l'émetteur **Particle_smoke.**

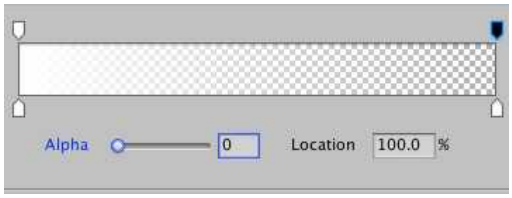
42.2. Dans *l'Inspector,* changez les paramètres :

Paramètres de base
42.2.1. ***Duration*** : 0.1
42.2.2. ***Looping*** : non
42.2.3. ***Start Speed*** : 6.7
42.2.4. ***Start Size*** : 1.25
42.2.5. ***Start Color*** : Blanc
42.2.6. ***Gravity Multiplier*** : 0.2
42.2.7. ***Inherit Velocity*** : 0
42.2.8. ***Simulation Space*** : Local
42.2.9. ***Play On Awake*** : non

Emission
42.2.10. ***Rate*** : 200

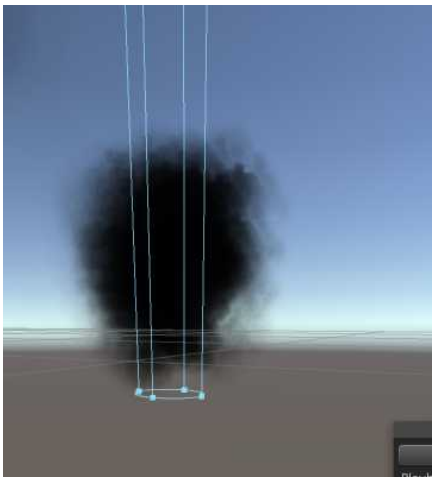
Shape
42.2.11. ***Shape*** **:** Cone
42.2.12. ***Angle*** **:** 2.0

42.2.13. ***Radius*** : 0.2



Color over Lifetime
42.2.14.***Couleur de début*** : Blanc, ***alpha***= 255
42.2.15.***Couleur de fin*** : Blanc, ***alpha***= 0

Renderer
42.2.16. ***Material*** : Recherchez **ParticleSmokeBlack**
42.2.17. ***Sort Mode*** : By Distance
42.2.18. ***Cast Shadows*** : non
42.2.19. ***Receive Shadows*** : non



Vous devriez avoir un effet de fumée pour accompagner les obstacles.

Sauvez votre scène (**CTRL+S**) ou (**⌘+S**) sur Mac.

Vous allez maintenant ajouter un autre effet, cette fois-ci pour représenter l'explosion des vases.

43. Dans la barre de menus, allez dans :

GameObject | Particles System

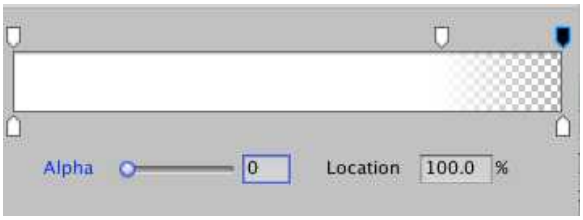
43.1. Nommez l'émetteur **Particle_vase.**

43.2. Dans *l'Inspector,* changez les paramètres :

Paramètres de base
43.2.1. ***Duration*** : 0.10
43.2.2. ***Looping*** : non
43.2.3. ***Start Lifetime*** : 1.37
43.2.4. ***Start Speed*** : 6.65
43.2.5. ***Start Size*** : 0.63
43.2.6. ***Start Color*** : (blanc)
43.2.7. ***Gravity Multiplier*** : 1
43.2.8. ***Inherit Velocity*** : 0
43.2.9. ***Simulation Space*** : Local
43.2.10. ***Play On Awake*** : non

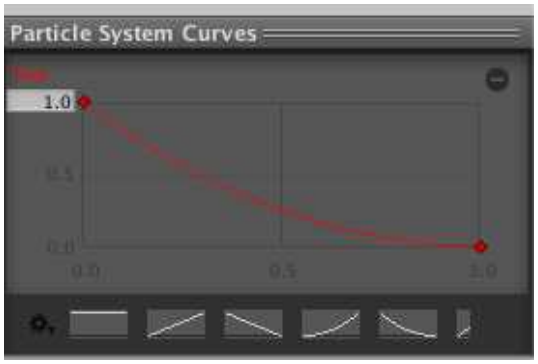
Emission
43.2.11. ***Rate*** : 124

Shape
43.2.12. ***Shape*** : Cone
43.2.13. ***Angle*** : 12
43.2.14. ***Radius*** : 0.2
43.2.15. ***Random Direction*** : yes



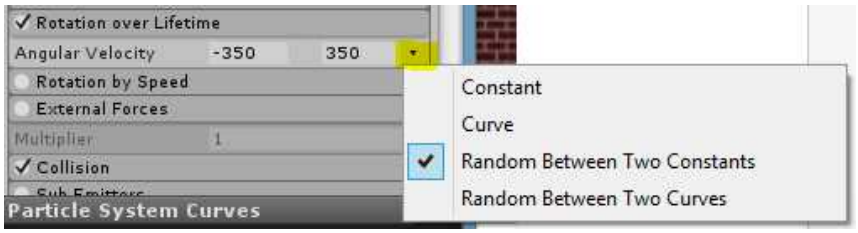
Color over Lifetime
43.2.16. ***Couleur de début*** = Blanc, ***alpha*** = 255
43.2.17. ***Couleur à 85%*** = Blanc, ***alpha*** = 255
43.2.18. ***Couleur de fin*** = Blanc, ***alpha*** = 0

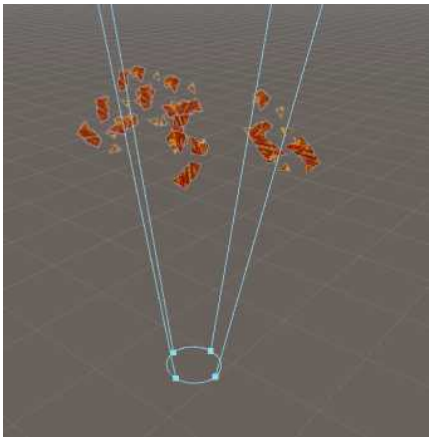
Size over Lifetime
43.2.19. ***Courbe*** : descendante



Rotation over Lifetime
43.2.20. ***Angular Velocity*** : Changez **Constant** pour **Random Between Two Constants.**
43.2.21. ***Angular Velocity*** : -350, 350

Pour changer de mode, cliquez sur la flèche à droite du paramètre :





Renderer
43.2.22. ***Material*** : Recherchez **Vase Break.**
43.2.23. ***Sort Mode*** : By Distance
43.2.24. ***Cast Shadows*** : non
43.2.25. ***Receive Shadows*** : non

Ajoutez maintenant une nouvelle lumière pour simuler l'incendie dans les fenêtres.

44. Dans la barre de menus, allez dans :

GameObject | Light | Spotlight

45. Changez les propriétés suivantes :

45.1. Nommez la lumière **Spotlight.**

45.2. Dans *l'Inspector*, changez les paramètres :

Transform
45.2.1. ***Position*** : **X**=4.18, **Y**=4.05, **Z**=2.63
45.2.2. ***Rotation*** **: X**=250**, Y**=0, **Z**=0
45.2.3. ***Type*** : Spot
45.2.4. ***Range*** : 103
45.2.5. ***SpotAngle*** : 179
45.2.6. ***Color*** : orange
45.2.7. ***Intensity*** : 2
45.2.8. ***Bounce Intensity*** : 0
45.2.9. ***Shadow Type*** : No Shadows

Vous devriez obtenir ce résultat :

Ajoutez une boule de débris en feu que le joueur devra évidemment éviter.

Récompenses et obstacles

46. Dans la barre de menus, allez dans :

GameObject | 3D | Sphere

47. À partir du panneau *Project*, glissez le matériel **lava** sur l'objet **Sphere** dans la scène.

48. Sélectionnez la **Sphere.**

48.1. Nommez-la **junk.**

48.2. Dans *l'Inspector,* changez les paramètres :

Mesh Renderer

48.2.1. ***Cast Shadows*** : Oui
48.2.2. ***Receive Shadows*** : non

Ajoutez un effet ***Halo*** sur **junk**.

49. Avec l'objet **junk** sélectionné, allez dans la barre de menus :

Component | Effects | Halo

49.1. Dans *l'Inspector,* changez :

Halo
49.1.1. ***Color*** : Orange
49.1.2. ***Size*** : 1

Vous devriez obtenir ce résultat :

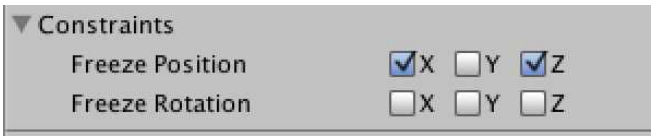
Maintenant, ajoutez un vase à attraper.

50. À partir de *Project,* glissez **vase_prefab** du répertoire **Prefab** vers *Hierarchy.*

50.1. Dans *l'Inspector,* changez les paramètres :

Transform
50.1.1. ***Position*** **: X=** 3.85, **Y=** 10.35, **Z=** -0.6

Rigidbody,
50.1.2. ***Constraints*** : **Freeze Position : X, Z**



Vous allez positionner l'objet **junk** dans la scène afin de la préparer pour la jouabilité.

51. Dans *Hierarchy*, sélectionnez **junk.**

51.1. Dans *l'Inspector,* changez :

Transform
51.1.1. ***Position*** **: X**=5.7, **Y**=12, **Z**=-0.6

Nous avons maintenant tous les éléments nécessaires pour la création du jeu. Dans le prochain chapitre, nous allons ajouter le code pour la jouabilité.

Chapitre 11 – Incendie : partie 2

Matière couverte dans ce chapitre

- **Ajout d'un script pour le déplacement du personnage**
- **Ajout d'un sol pour les collisions**
- **Ajout de scripts pour les vases et les débris**
- **Ajout d'une interface (UI)**
- **Ajout de règles au jeu**
- **Ajout d'une défaite**
- **Ajout d'un *Light Probe Group***
- **Activation du pointage**
- **Ajout de sons**
- **Ajout d'une ambiance visuelle**

Dans ce chapitre, vous allez compléter l'exercice du chapitre précédent en y ajoutant les scripts qui vont contrôler la jouabilité.

Ouvrez le projet « Incendie » et la scène « game ».

Atelier pratique

Programmation du jeu

Vous allez ajouter un script qui gère le déplacement du personnage avec le mouvement de la souris.

Ajout d'un script pour le déplacement du personnage

1. À partir de *Project*, glissez le script **character.js** qui se trouve dans le répertoire **Scripts** sur l'objet **perso.**

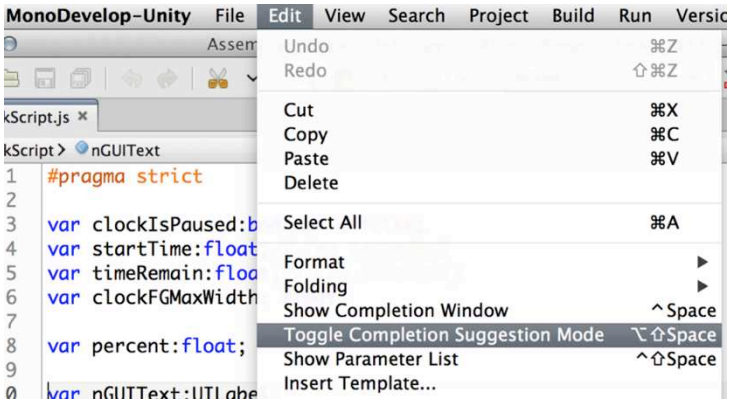
Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Ouvrez ce script avec *MonoDevelop* ou un autre éditeur de script.

Si vous utilisez *MonoDevelop*, pensez à désactiver l'auto-remplissage des commandes (***Toggle Completion Suggestion Mode***).

2. À partir de la barre de menus *MonoDevelop,* appuyez sur :

Edit | Toggle Completion Suggestion Mode.



Ajoutez ce bloc de codes :

```
private var anim:Animator;
var speed:float = 30;

function Awake(){
    anim = GetComponent.<Animator>();
}

function Update () {
    var trans_X = Input.GetAxis("Mouse X") * speed;
    trans_X = trans_X * Time.deltaTime;

    anim.SetFloat("horizontal", Input.GetAxis("Mouse X"));
    transform.Translate (-trans_X, 0,0);
    if (transform.position.x > 7.5){
        transform.position.x = 7.5;
    }
    if (transform.position.x < 0){
        transform.position.x = 0;
    }
}
```

↘ *Les textes en caractères **gras** sont à ajouter, le reste demeure inchangé.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Au début du programme (**Awake()**), ce script récupère le Component (**<Animator>()**) et garde une référence dans la variable (**anim**).

Continuellement pendant le jeu (Update), il faut regarder la valeur de déplacement de la souris horizontalement (**Input.GetAxis("Mouse X")**). On multiplie cette valeur par la vitesse (* speed) et on multiplie le produit par le ***deltaTime***. Finalement, on conserve cette valeur pour la translation du personnage (**trans_X**).

Le script va ensuite changer la valeur du paramètre (**"horizontal"**) qui se trouve dans l'*Animator Controller* (**anim.SetFloat**) en lui assignant la valeur de déplacement (**Input.GetAxis("Mouse X")**).

Notez-bien!

*Le **deltaTime** permet de garder le déplacement constant indépendamment de la vitesse de déroulement du jeu.*

Notez-bien!

*C'est la valeur du paramètre **horizontal** qui permet l'animation du personnage à gauche, à droite ou sur place.*

On poursuit avec une translation du personnage (**transform.Translate**) dans la direction (**trans_X**).

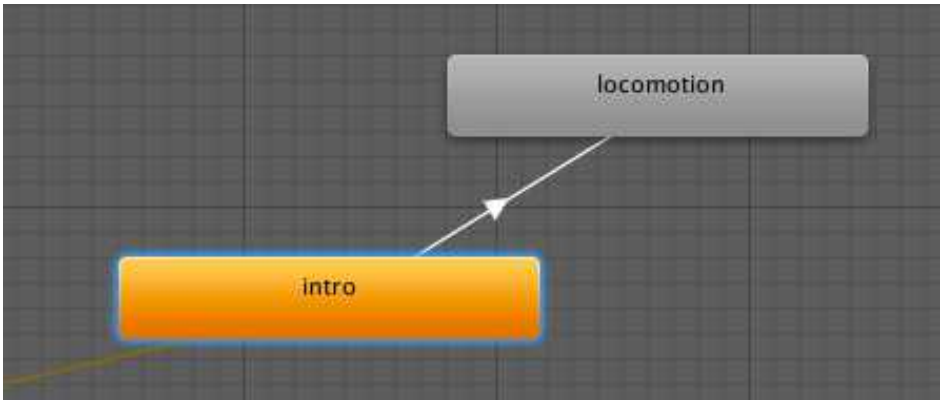
On regarde ensuite si la position du personnage dépasse les limites permises (sa **Position X** ne doit pas être **< 0** ou **> 7.5**). Si c'est le cas, on replace le personnage à cette frontière (**transform.position.x**).

Testez votre jeu.

Vous devriez être en mesure de déplacer le personnage. Attention, celui-ci ne devrait pas pouvoir se déplacer durant l'**intro**.

Retournez en mode édition.

Vous allez désormais modifier le script pour limiter son déplacement afin qu'il ne puisse pas bouger durant l'**intro**, mais seulement durant l'animation **locomotion.**



Ouvrez le script « character.js ».

Ajoutez les lignes suivantes :

```
static var currentBaseState:AnimatorStateInfo;
private var anim:Animator;
var speed:float = 30;
var lockmouse:boolean = true;

function Awake(){
  anim = GetComponent.<Animator>();
}

function Update () {
  currentBaseState = anim.GetCurrentAnimatorStateInfo(0);
  if (!lockmouse){
     var trans_X = Input.GetAxis("Mouse X") * speed;
     trans_X = trans_X * Time.deltaTime;
     anim.SetFloat("horizontal", Input.GetAxis("Mouse X"));
     transform.Translate (-trans_X, 0,0);
     if (transform.position.x > 7.5){
        transform.position.x = 7.5;
     }
     if (transform.position.x < 0){
        transform.position.x = 0;
     }
  }else if (currentBaseState.IsName("Base Layer.locomotion")){
     lockmouse = false;
  }
}
```

↘ *Les textes en caractères **gras** sont à ajouter, le reste demeure inchangé.*

Sauvegardez le script et retournez dans l'éditeur.

En détails…

Au lancement du jeu, on bloque le déplacement avec la variable (lockmouse).

Ces modifications font qu'à partir de maintenant et tout au long du déroulement du jeu, le script récupère l'état de l'***Animator*** (**anim.GetCurrentAnimatorStateInfo(0)**) et place cette valeur dans une variable (**currentBaseState**).

Notez-bien!

On utilisera cette valeur pour connaître le moment propice afin de déverrouiller le déplacement du joueur.

On regarde ensuite la valeur de (**lockmouse**). Si elle est vraie, on vérifie si l'état actuel est nommé **locomotion** (**IsName ("Base Layer.locomotion")**). Si c'est le cas, l'intro est complété. On débloque alors le déplacement (**lockmouse = false**) permettant ainsi d'utiliser le code écrit préalablement.

Testez votre jeu.

Présentement, l'animation d'**intro** joue, le vase passe au travers du personnage. Après l'animation, il peut être contrôlé avec la souris.

Retournez en mode édition.

Ajoutez un ***Collider*** au personnage ainsi qu'un sol afin que les objets puissent entrer en contact avec ces éléments.

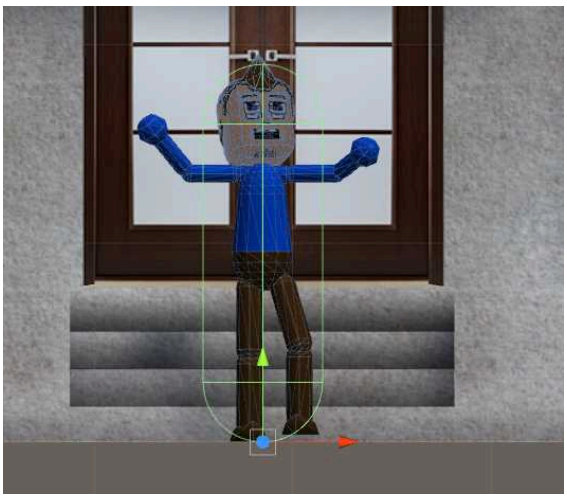
3. Dans *Hierarchy*, sélectionnez l'objet **perso.**

3.1. Dans *l'Inspector,* appuyez sur le bouton « Add Component », puis :

Physics | Capsule Collider

3.2. Changez les paramètres suivants :

Capsule Collider
3.2.1. ***Center*** : **X=**0, **Y=**0, **Z=**0.95
3.2.2. ***Radius*** : 0.3
3.2.3. ***Height*** : 1.9
3.2.4. ***Direction*** **:** Z-Axis

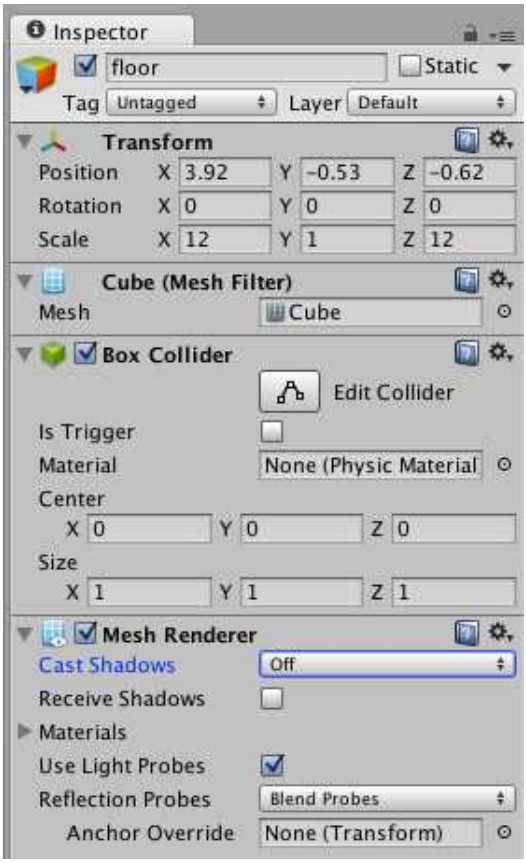


Vous allez ajouter un sol qui permettra d'entrer en collision avec les objets échappés.

Ajout d'un sol

4. Dans la barre de menus, allez dans :

GameObject | 3D Object | Cube



4.1. Nommez-le **floor.**

4.2. Dans *l'Inspector*, changez les paramètres :

Transform
4.2.1.***Position*** : **X=** 3.92, **Y=** -0.5, **Z=** -0.62
4.2.2.***Scale*** : **X=** 12, **Y=** 1, **Z=** 12

Mesh Renderer
4.2.3.***Cast Shadows*** : non
4.2.4.***Receive Shadows*** : non

Les objets vont maintenant percuter le sol et le personnage. Il nous faut un moyen rapide d'identifier avec quels éléments les objets entrent en contact.

5. À partir de la barre de menus, allez dans :

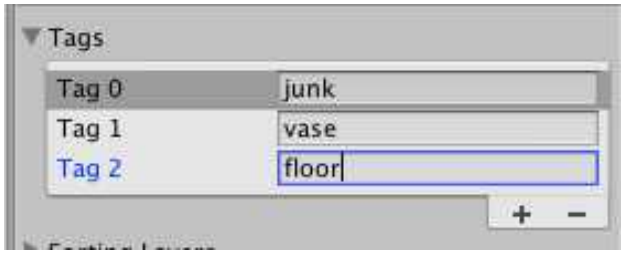
Edit | Project Settings | Tags & Layers

Notez-bien!

*La section **Tags** est cachée par défaut, vous devez appuyer sur la flèche pour voir ses propriétés.*

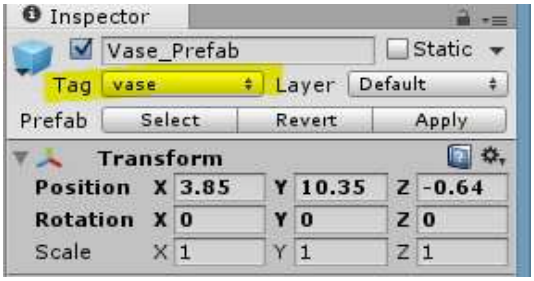
6. Ajoutez les ***Tags*** suivants en appuyant sur le bouton **(+)**.

- ***Tag 0*** **:** junk
- ***Tag 1*** **:** vase
- ***Tag 2*** **:** floor

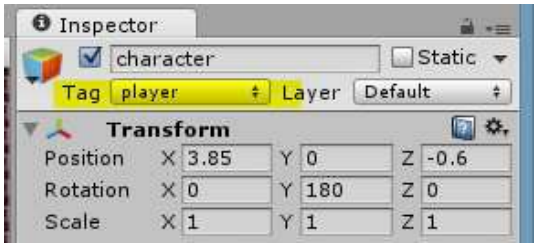


7. Dans *l'Inspector*, appliquez les ***Tags*** suivants sur les objets de *Hierarchy* :

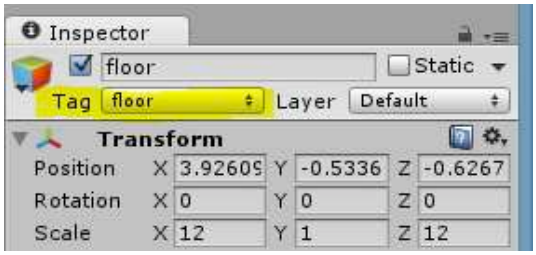
7.1. Sur l'objet : **junk** changez le ***Tag*** pour **junk.**



7.2. Sur l'objet : **Vase_Prefab** changez le ***Tag*** pour **vase.**



7.3.Sur l'objet : **character** changez le ***Tag*** pour **Player.**



7.4.Sur l'objet : **floor** changez le ***Tag*** pour **floor.**

Les ***Tags*** sont appliqués, il faut ajouter le code pour le vase.

Ajout d'un script pour les vases

Chute libre

La première opération à effectuer sera de replacer le vase dans le ciel lorsqu'une collision est détectée.

1. À partir de *Project*, glissez le script **vase.js** du répertoire **Scripts** sur l'objet **Vase_Prefab**.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Ouvrez le script « vase.js ».

Ajoutez le code suivant :

```
private var origins:Vector3;

function Start(){
    origins = transform.position;
}

function OnCollisionEnter(col:Collision){
    replaceObj();
}

function replaceObj(){
    transform.rotation = Quaternion.Euler(Vector3.zero);
    transform.position = origins;
    GetComponent.<Rigidbody>().velocity = Vector3.zero;
}
```

↘ *Les textes en caractères **gras** sont à ajouter, le reste demeure inchangé.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

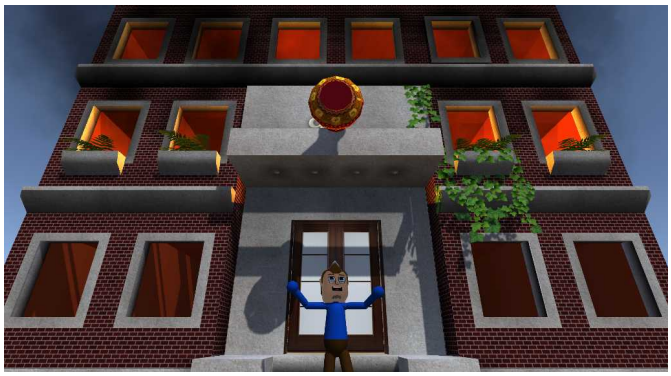
Au début du jeu (**Start()**), on mémorise dans une variable (**origins**) la position d'origine du **vase** pour des références futures.

Quand le **vase** entre en collision avec un élément (**OnCollisionEnter()**), on appelle cette fonction (**replaceObj()**). Elle réinitialise la rotation (**transform.rotation**).

(**transform.position**) est la fonction de la position et (**GetComponent.<Rigidbody>().velocity**) celle de la vélocité du **vase**.

Testez votre jeu.

Des vases devraient maintenant tomber du ciel continuellement. Malheureusement, ils seront toujours alignés sur la même trajectoire. Il n'y aura donc aucun challenge.



Retournez en mode édition.

Position aléatoire

Vous allez ajouter des effets de hasard à la position d'origine du vase, sur l'axe horizontal.

Ouvrez le script « vase.js ».

Changez la ligne suivante :

```
private var origins:Vector3;

function Start(){
    origins = transform.position;
}

function OnCollisionEnter(col:Collision){
    replaceObj();
}

function replaceObj(){
    transform.rotation = Quaternion.Euler(Vector3.zero);
    origins.x = Random.Range(0.5, 7.0);
    transform.position = origins;
    GetComponent.<Rigidbody>().velocity = Vector3.zero;
}
```

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Au début du jeu (**Start()**), le script mémorise la position d'origine du **vase** pour de futures références comme nous l'avons mentionné antérieurement. Par contre, avant de repositionner le vase, on choisit au hasard une nouvelle valeur pour l'axe X (**origins.x = Random.Range**) entre **0.5** et **7.0.**

Testez votre jeu.

Des vases devraient maintenant tomber à des positions aléatoires.

Retournez en mode édition.

Ajouts de particules

Dans le chapitre précédent, vous avez créez deux effets de particules pour le vase :

• **Les étoiles,** quand on attrape le vase.
• **Les débris,** quand on l'échappe.

La prochaine étape consistera à intégrer ces deux effets.

Ouvrez le script « vase.js ».

Ajoutez les lignes suivantes :

```
var catchParticles:ParticleSystem;
var missParticles:ParticleSystem;
private var origins:Vector3;

…

function OnCollisionEnter(col:Collision){
   var objCopy:ParticleSystem;
   var particlesSelect:ParticleSystem;

   if (col.gameObject.tag == "Player"){
      particlesSelect = catchParticles;
   }else if (col.gameObject.tag == "junk"){
      replaceObj();
      return;
   }else
      particlesSelect = missParticles;
   objCopy = Instantiate(particlesSelect,
   transform.position,
   Quaternion.identity);
   objCopy.Play();
   Destroy(objCopy.gameObject, 2);
   replaceObj();
}
…
```

↘ *Les textes en **gras** sont à ajouter et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails…

Vous venez de déclarer deux variables pour les émetteurs de particules (catchParticles et missParticles).

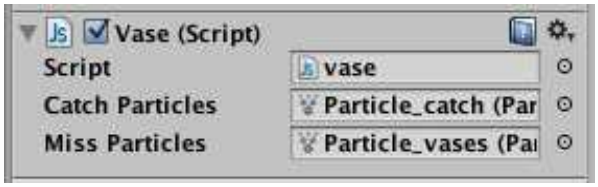
Quand une collision est détectée, le script interroge le ***Tag*** sur l'objet en contact avec le vase.

Si le Tag est « Player » (**(col.gameObject.tag == "Player")**) alors, on assigne les particules (**catchParticules**) en tant que particules sélectionnées (**particlesSelect**). Si le ***Tag*** est « junk », on replace le vase et on sort de la fonction avec la commande (***return***). Sinon, on assigne (**missParticles**) dans (**particlesSelect**).

Ensuite, on fait une copie de l'émetteur sélectionné (**Instantiate…**) et on émet quelques particules (**objCopy.Play()**). Après un laps de 2 secondes, on retire le nouvel émetteur de la scène (**Destroy(objCopy.gameObject, 2)**).

2. Sélectionnez l'objet **Vase_Prefab** dans *Hierarchy.*

2.1. Dans *l'Inspector*, changez :



Vase
2.1.1. ***Catch Particles*** : Glissez l'objet : **Particle_catch** de *Hierarchy*.
2.1.2. ***Miss Particles*** : Glissez l'objet : **Particle_vase** de *Hierarchy*.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu

Retournez en mode édition.

Les effets de particules pour le vase devraient êtres bons, mais le personnage ne tente pas de l'attraper. Il ne fait que le regarder tomber. Il faut donc déclencher l'animation **catch** durant l'attrapée.

Ouvrez le script « character.js ».

Ajoutez la fonction suivante :

```
…
function catchObj(){
    anim.SetTrigger("catch");
}
```

↘ *Les textes en caractères **gras** sont à ajouter, et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script

Ouvrez le script « vase.js ».

Ajoutez la ligne suivante :

```
…

function OnCollisionEnter(col:Collision){
  var objCopy:ParticleSystem;
  var particlesSelect:ParticleSystem;

  if (col.gameObject.tag == "Player"){
     col.gameObject.SendMessage("catchObj");
     particlesSelect = catchParticles;
  }else if (col.gameObject.tag == "junk"){
     replaceObj();
     return;
  }else
     particlesSelect = missParticles;
  objCopy = Instantiate(particlesSelect,transform.position,
                                          Quaternion.identity);
  objCopy.Play();
  Destroy(objCopy.gameObject, 2);
  replaceObj();
}
…
```

↘ *Les textes en **gras** sont à ajouter et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Quand le **vase** détecte une collision avec le personnage (**col.gameObject.tag == "Player"**), on appelle la fonction (**catchObj()**) que nous venons d'ajouter dans le script (**character.js**).

La fonction (**catchObj()**) déclenche le paramètre « catch » dans l'*Animator Controller.*

Testez le jeu

Le personnage devrait maintenant faire l'action d'attraper quand le vase entre en collision avec lui.

Retournez en mode édition.

La prochaine étape sera de modifier la boule de feu pour qu'elle agisse de façon similaire.

Ajout d'un script pour les débris

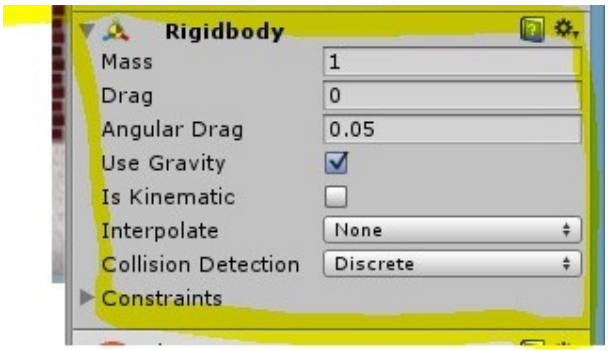
Gravité

Vous avez probablement remarqué que la boule de feu ne bouge pas. Il faut ajouter un ***Rigidbody*** pour activer la gravité.

3. Sélectionnez l'objet **junk** dans *Hierarchy*.

3.1. Dans *l'Inspector,* appuyez sur le bouton « Add Component », puis :

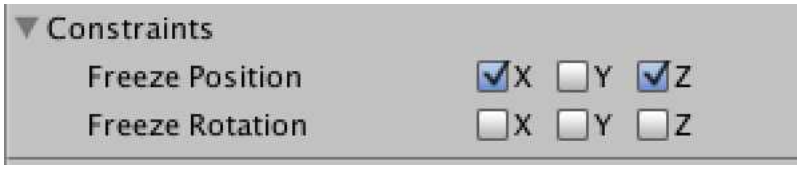
Physics | Rigidbody



Vous allez également limiter les déplacements sur l'axe Y afin qu'ils soient toujours alignés avec le personnage.

3.2. Changez les ***Constraints*** suivantes:

Rigidbody
3.2.1.***Freeze Position*** **: X=** oui, **Y=** non, **Z=** oui



4. À partir de *Project*, glissez le script **junk.js** du répertoire **Scripts** sur l'objet **junk**.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Ouvrez le script « junk.js ».

Ajoutez les codes suivants :

```
private var origins:Vector3;
var smokeball:ParticleSystem;

function Start () {
    origins = transform.position;
}

function OnCollisionEnter(col:Collision){
    if (col.gameObject.tag != "vase"){
        var objCopy = Instantiate(smokeball, transform.position,
                                  Quaternion.identity);
        objCopy.Play();
        Destroy(objCopy.gameObject, 2);
        replaceObj();
    }
}

function replaceObj(){
    transform.rotation = Quaternion.Euler(Vector3.zero);
    origins.x = Random.Range(0.5, 7.0);
    transform.position = origins;
    GetComponent.<Rigidbody>().velocity = Vector3.zero;
}
```

↘ *Les textes en caractères **gras** sont à ajouter, le reste demeure inchangé*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Ce script fait la même action que le **vase** mais sans les particules de récompenses, seulement une fumée (**smokeball**).

5. Sélectionnez l'objet **junk.**

5.1. Dans l'Inspector, changez :

Junk (Script)

5.1.1. ***Smokeball*** : Glissez l'élément **Particle_smoke** de *Hierarchy*.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu

Retournez en mode édition.

Le déroulement du jeu devrait être complet, si on lui ajoute quelques éléments :

- **Une interface utilisateur**
- **Des règlements (une fin, un pointage…)**
- **Des expressions au joueur avec ses animations**
- **Des effets sonores**
- **Des filtres de lentilles**

Ajout d'une interface

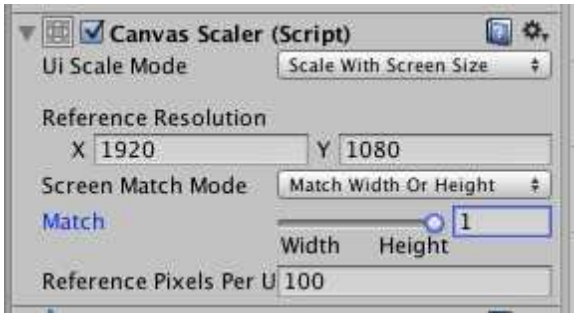
Création d'un *Canvas*

1. À partir de la barre de menus, allez dans :

GameObject | UI | Canvas

1.1. Nommez-le **CanvasUI.**

1.2. Dans *l'Inspector*, changez :



Canvas Scaler
1.2.1. ***UI Scale Mode***: Scale With Screen Size
1.2.2. ***Reference Resolution*** **: X=**1920, **Y=**1080
1.2.3. ***Screen Match Mode*** : Match Width Or Height
1.2.4. ***Match*** : 1

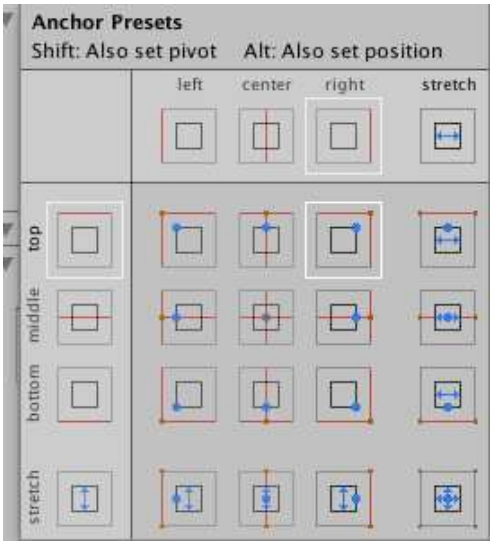
Texte pour les vases attrapés

2. Sélectionnez **canvasUI**, allez dans la barre de menus, puis :

GameObject | UI | Text

2.1. Nommez-le **GUI Catch.**

2.2. Dans *l'Inspector,* changez les paramètres suivants :



Rect Transform
2.2.1. ***Anchor Presets*** : **top | right**.

2.2.2. ***Pox*** **X=** -20, ***Pos*** **Y=** -20, ***Pos*** **Z=** 0
(Il est préférable de garder ces paramètres pour la fin).

2.2.3. ***Width*** **:** 400, ***Height*** **:** 120

Anchors
2.2.4. ***Min*** **: X=** 1, **Y=** 1
2.2.5. ***Max*** **: X=** 1, **Y=** 1
2.2.6. ***Pivot*** **: X=** 1, **Y=** 1

Text (Script)
2.2.7. ***Text*** : Catch :
2.2.8. **Color** : Blanc

Character
2.2.9 ***Font*** : Recherchez **CharisSILMali-B** dans *Project.*
2.2.10 ***Font Style*** : Normal
2.2.11 ***Font Size*** : 96
2.2.12 ***Line Spacing*** : 1
2.2.13 ***Rich Text*** *:* oui

Paragraph
2.2.14. ***Alignment*** : Droite, Milieu

2.2.15. ***Horizontal Overflow*** : Overflow
2.2.16. ***Vertical Overflow*** : Overflow
2.2.17. ***Best Fit*** : non
2.2.18. ***Color*** : Blanc
2.2.19. ***Material*** : Aucun

Texte pour les vases ratés

Vous allez créer un 2e champ texte avec le même alignement et quelques pixels de décalage sur l'axe-Y.

3. Sélectionnez **GUI Catch** dans *Hierarchy*.

3.1. Créez un doublon de **GUI Catch,** en appuyant sur **(CTRL+D)** ou **(⌘+D)** sur Mac.

3.2. Nommez-le **GUI Miss.**

3.3. Dans *l'Inspector*, changez les paramètres suivants :

Rect Transform
3.3.1. ***Pox*** **X=** -20, ***Pos*** **Y=** -140, ***Pos*** **Z=** 0

Text (Script)
3.3.2. ***Text*** : Miss :
3.3.3. ***Color*** : Orange

Texte pour la fin du jeu

Vous allez ajouter un 3e **texte**. Cette fois-ci, pour annoncer le message que la partie est terminée.

4. À partir de la barre de menus, allez dans :

GameObject | UI | Text

4.1. Nommez-le **GUI GameOver.**

4.2. Dans *l'Inspector,* changez les paramètres suivants :

Rect Transform
4.2.1. ***Pox*** **X =** 0, ***Pos*** **Y =** 0, ***Pos*** **Z =** 0
(Il est préférable de garder ces paramètres pour la fin).

4.2.2. ***Width*** **:** 400, ***Height*** **:** 120

Anchors
4.2.3. ***Min* : X=** 0.5**, Y=** 0.5
4.2.4. ***Max* : X=** 0.5**, Y=** 0.5
4.2.5. ***Pivot* : X=** 0.5**, Y=** 0.5

Text (Script)
4.2.6. ***Text*** : GAME OVER

Character
4.2.7. ***Font*** : CharisSILMali-B
4.2.8. ***Font Style*** : Normal
4.2.9. ***Font Size*** : 96
4.2.10. ***Line Spacing*** : 1
4.2.11. ***Rich Text*** : oui

Paragraph
4.2.12. ***Alignment*** : Centré, Milieu
4.2.13. ***Horizontal Overflow*** : Overflow
4.2.14. ***Vertical Overflow*** : Overflow
4.2.15. ***Best Fit*** : non
4.2.16. ***Color*** : Rouge
4.2.17. ***Material*** : Aucun

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Bouton pour rejouer

Il manque encore un bouton pour réinitialiser la scène quand la partie est terminée.

5. À partir de la barre de menus, allez dans :

GameObject | UI | Bouton

6. Sélectionnez le nouveau bouton.

6.1. Nommez-le **Btn_retry.**

6.2. Dans *l'Inspector,* changez :

Rect Transform
6.2.1. ***Pox* X=** 0**, *Pos* Y=** -150**, *Pos* Z=** 0
6.2.2. ***Width*** : 250, ***Height*** : 60

Button (Script)
6.2.3. ***Highlighted Color*** : Jaune

7. Dans *Hierarchy*, à l'intérieur de ce bouton, sélectionnez **Text.**

7.1. Dans *l'Inspector*, changez :

Text (Script)
7.1.1. ***Text* :** Rejouer?
7.1.2. ***Font* :** CharisSILMali-B
7.1.3. ***Font Size* :** 35
7.1.4. ***Horizontal Overflow* :** Overflow
7.1.5. ***Vertical Overflow* :** Overflow

Vous devriez obtenir ce résultat :

Les éléments du jeu sont en place, vous allez vous attaquer au ***script*** pour la jouabilité.

Ajout du gameplay

Il faut ajouter un script pour le gameplay de la scène. Comme vous avez présentement un canevas, un objet nommé **EventSystem** se retrouve automatiquement dans *Hierarchy*, vous l'utiliserez pour vos besoins.

1. À partir de *Project*, glissez le ***script* game.js** sur l'objet **EventSystem** dans *Hierarchy*.

Débutez simplement en faisant disparaître le bouton ainsi que le texte de la fin de la partie.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Ouvrez le script « game.js ».

Ajoutez les codes suivants :

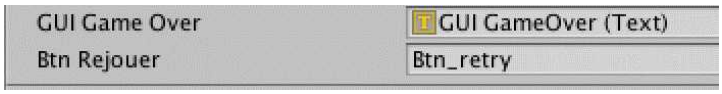
```
var GUIGameOver:UI.Text;
var BtnRejouer:GameObject;
private var gameover = false;

function Start(){
    BtnRejouer.SetActive(false);
    gameover = false;
    GUIGameOver.text = "";
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé*

Sauvegardez le script et retournez dans l'éditeur.

2. Sélectionnez l'objet **EventSystem** dans *Hierarchy*.

2.1. Dans *l'Inspector*, changez :



Game (Script)
2.1.1.***GUI Game Over*** : GUI GameOver
2.1.2.***Btn Rejouer*** : Btn_retry

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu

Les deux objets devraient être invisibles lors du déroulement du jeu.

Retournez en mode édition.

Ajout d'une défaite

Continuez avec les mêmes objets. Maintenant que le bouton ainsi que le texte sont cachés, vous allez ajouter le code nécessaire pour les rendre actifs.

Ouvrez le script « game.js ».

Ajoutez le code suivant :

```
…
private var gameover = false;
var playerObj:GameObject;

function Start () {
  BtnRejouer.SetActive(false);
  gameover = false;
  GUIGameOver.text = "";
}

function hitPlayer(){
  GUIGameOver.text = "GAME OVER";
  gameover = true;
  playerObj.SetActive(false);
  BtnRejouer.SetActive(true);
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé*

Sauvegardez le script.

Ouvrez le script « junk.js ».

Ajoutez les lignes suivantes :

```
var scriptObj:GameObject;
var smokeball:ParticleSystem;
…
function OnCollisionEnter(col:Collision){
 if (col.gameObject.tag == "Player")
    scriptObj.SendMessage("hitPlayer");

 if (col.gameObject.tag != "vase"){
    var objCopy = Instantiate(smokeball,transform.position,
                              Quaternion.identity);
    objCopy.Play();
    Destroy(objCopy.gameObject, 2);
    replaceObj();
 }
}
…
```

↘ *Les textes en caractères **gras** sont à ajouter, et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Dans le script **game.js,** vous venez d'ajouter une fonction de fin de jeu (**hitPlayer()**). Celle-ci remplit le texte (**GUIGameOver.text**), fait disparaître le personnage (**playerObj.SetActive (false)**) et apparaître le bouton (**BtnRejouer.SetActive(true)**).

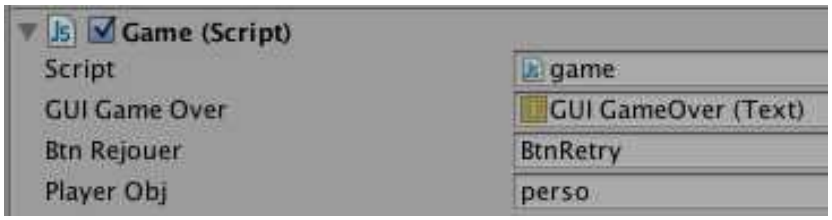
Le script junk.js appelle la fonction précédente (**SendMessage("hitPlayer")**) quand la boule de feu entre en contact avec le joueur (**col.gameObject.tag == "Player"**).

Il ne vous reste qu'à assigner les éléments dans les différentes variables.

3. Sélectionnez l'objet **EventSystem** dans *Hierarchy*.

3.1. Dans *l'Inspector,* changez :

Game (Script)
3.1.1. ***Player Obj***: Glissez votre **perso** de *Hierarchy.*



4. Sélectionnez l'objet **junk** dans *Hierarchy*.

4.1. Dans *l'Inspector,* changez :

Junk (Script)
4.1.1. ***Script Obj***: Glissez **EventSystem** de *Hierarchy.*

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu

Vous devriez obtenir le résultat ci-dessous quand la boule de feu touche le joueur. Par contre le bouton « **Rejouer ?** » ne fonctionne pas pour l'instant.

Retournez en mode édition.

Ajoutez une fonction pour le bouton dans le script **game.js.**

Ouvrez le script « game.js ».

Ajoutez la fonction suivante :

```
…
function hitPlayer(){
 GUIGameOver.text = "GAME OVER";
 gameover = true;
 playerObj.SetActive(false);
 BtnRejouer.SetActive(true);
}

function restart(){
 Application.LoadLevel(Application.loadedLevel);
}
```

↘ *Les textes en caractères **gras** sont à ajouter, et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

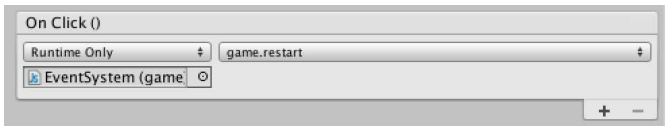
Il faut indiquer au bouton **Btn_retry** d'appeler la fonction **restart()** quand on appuie dessus.

5. Sélectionnez le bouton **Btn_retry** qui se trouve dans le **CanvasUI.**

5.1. Dans la section **On Click ()**, appuyez sur le bouton **(+)** pour ajouter une fonction.

5.1.1. Glissez **EventSystem** dans le champ du bas.
5.1.2. Dans la case de droite, sélectionnez la fonction suivante :

Game | restart()



Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu.

Le bouton devrait être fonctionnel à 100%. Il est également possible qu'il y ait un effet secondaire à cette **Application.LoadLevel()**. Si la scène est complètement assombrie comme dans l'exemple ci-dessous, voici comment régler le problème.

Retournez en mode édition.

6. Dans la barre de menus, allez dans :

Window | Lighting

6.1. Dans le panneau *Lighting*, section ***Scene,*** changez :

Other Settings
6.1.1. Décochez ***Continuous Baking***, car celui-ci cause le problème.

6.1.2. Appuyez sur « Build » pour préparer les ***lightmaps.***
Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu.

L'unique problème avec les ***lightmaps***, c'est que ceux-ci ne fonctionnent pas avec le ***Skin Mesh Renderer***. Notre personnage ne réagit pas bien à une scène qui utilise des ***lightmaps*** pré-calculés.

Le résultat produit est un visage complètement noir.

Retournez en mode édition.

Pour fixer ce problème, vous allez utiliser des ***Light Probe Group***.

« Light Probe Group »

Les ***lights probes groups*** sont des zones de couleur. Elles servent à dicter la réaction de la lumière avec les éléments qui ne peuvent pas utiliser des ***lightmaps*** pré-calculés : le visage de notre personnage.

L'avantage d'utiliser les ***Lights Probes Groups***, c'est qu'ils permettent de réduire significativement les besoins en ressources des appareils. Ils améliorent également les performances et la qualité visuelle du jeu.

Vous devez donc couvrir la partie visible de la scène avec des boules de couleur.

Création du Light Probe Group

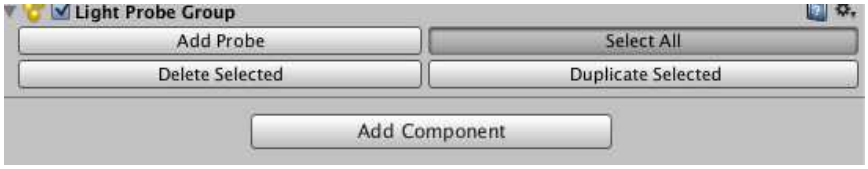
1. À partir de la barre de menus, allez à :

GameObject | Light | Light Probe Group

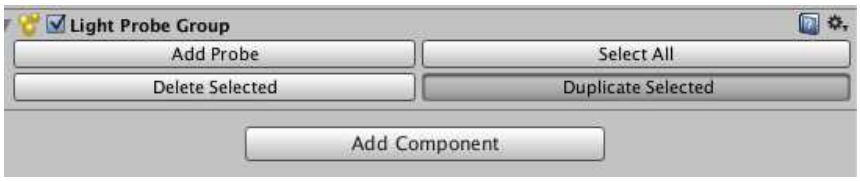
Quatre boules jaunes liées par des lignes roses devraient apparaître dans la fenêtre *Scene.*

2. Dans *Hierarchy*, sélectionnez **Light Probe Group.**

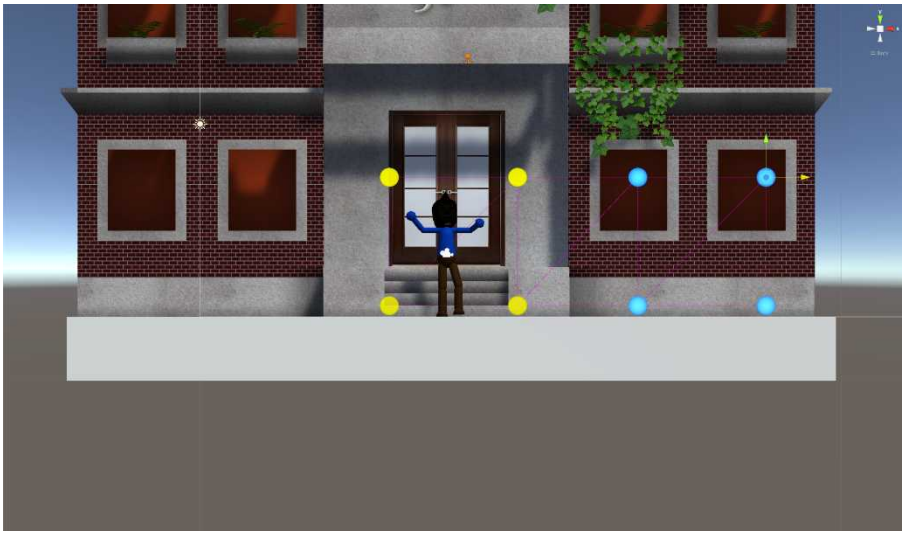
2.1. Dans *l'Inspector*, appuyez sur le bouton « Select All ».



2.2. Appuyez ensuite sur « Duplicate Selected » pour créer plus de ***Probes.***

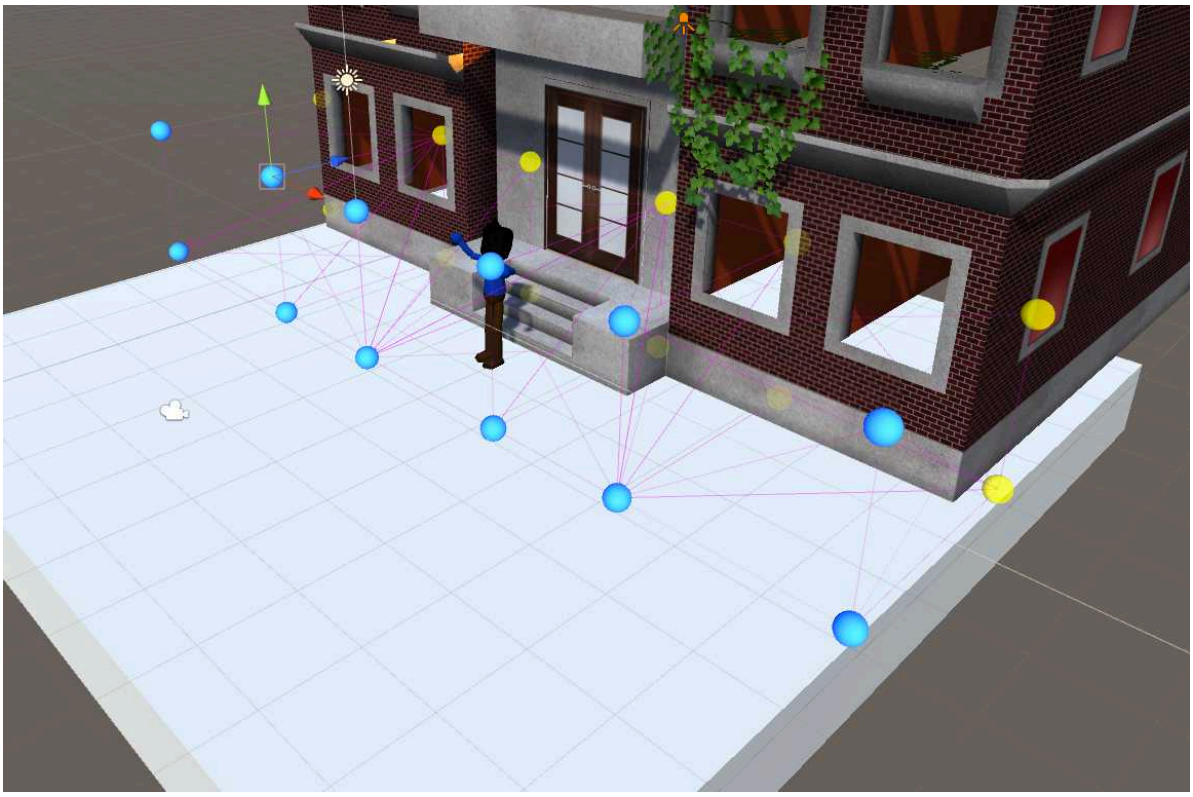


2.3. Dans votre *Scene*, repositionnez ces nouveaux ***Probes*** à égales distances des boules originales.



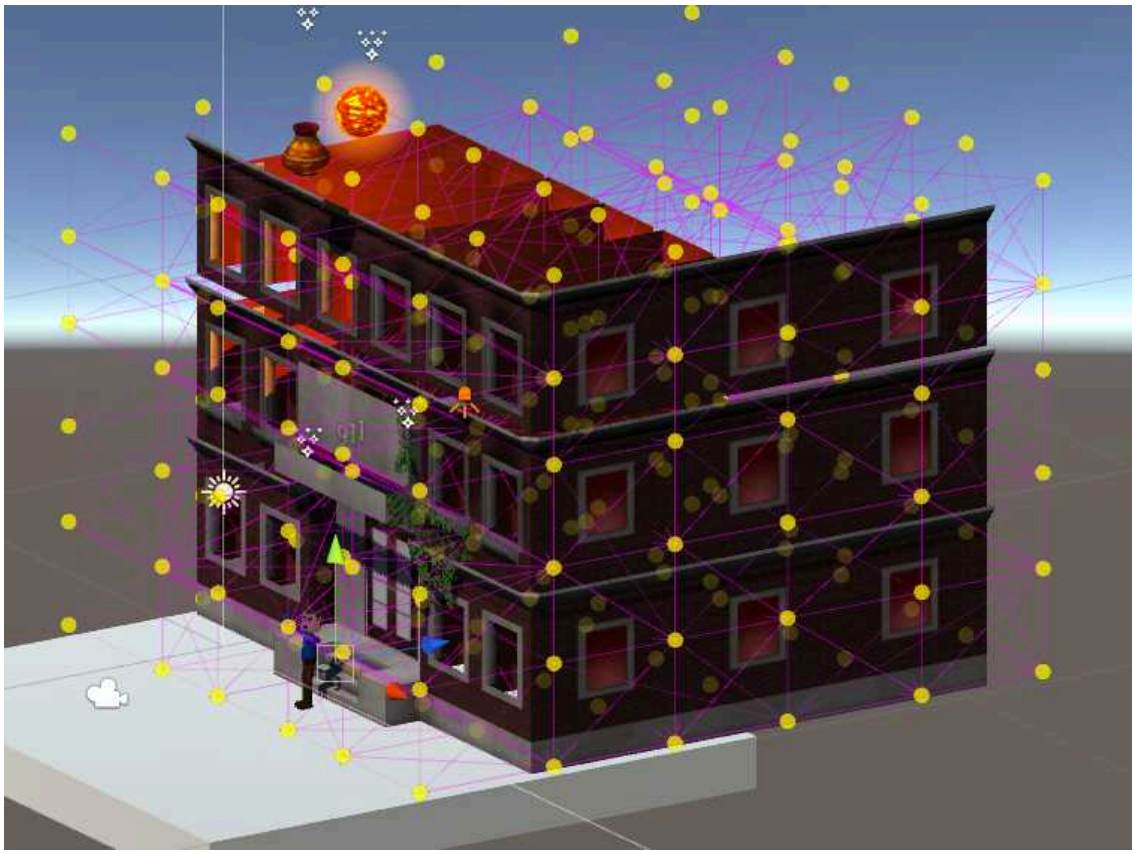
2.4. Refaites quelques « Duplicate Selected » pour couvrir la largeur de la surface de jeu.

2.5. Repositionnez également les **Probes** afin de bien couvrir votre personnage.



2.6. Dupliquez et répétez ces étapes afin de couvrir entièrement votre scène.

Tentez de reproduire ce résultat :



Le ***Light Probe Group*** est maintenant construit. Il faut indiquer aux lumières quels rôles elles vont jouer dans le calcul des ***Lightmaps.***

Préparation des lumières

3. Dans *Hierarchy*, sélectionnez **Directional Light.**

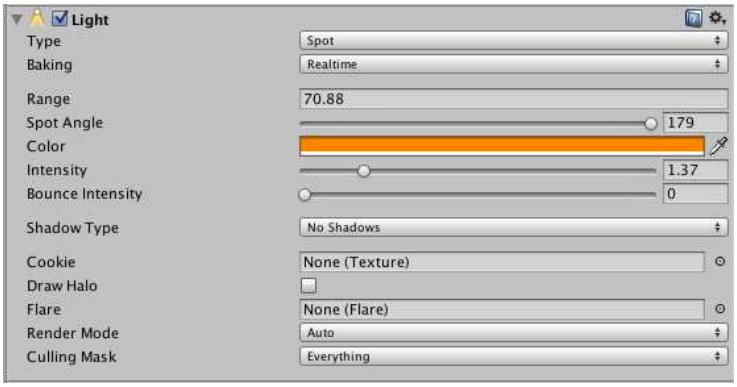
3.1. Dans *l'Inspector*, changez :

Light
3.1.1. **Baking** : Mixed

Cette lumière est en mesure d'interagir avec le calcul des Lightmaps.

4. Dans *Hierarchy*, sélectionnez **Spotlight.**

4.1. Dans *l'Inspector*, changez :



Light
4.1.1. ***Baking*** : Realtime

Cette lumière n'influencera pas le ***Lightmap,*** mais elle sera toujours active en temps réel.

Configuration des objets de la scène

5. Dans *Hierarchy*, sélectionnez ***les enfants*** de **tete** dans l'objet **perso.**

Ces éléments se nomment : **tete_default, tete_happy, tete_sad, tete_stressed** et **tete_suprised.**

5.1. Dans *l'Inspector*, changez :

Mesh Renderer
5.1.1. ***Use Light Probes*** : oui

Faites le même déroulement pour le **Vase_Prefab.**

6. Dans *Hierarchy*, sélectionnez l'objet **Vase_Prefab.**

6.1. Dans *l'Inspector*, changez :

Mesh Renderer
6.1.1. ***Use Light Probes*** : oui

Pour ajouter **building**

7. Dans *Hierarchy*, sélectionnez l'objet **Building** qui est l'enfant de **building.**

8.1. Dans *l'Inspector*, changez :

Mesh Renderer
8.1.1. ***Use Light Probes*** : oui

Ce **building** sera désormais illuminé par le ***lightmap*** et la lumière en temps réel (**Spotlight**).

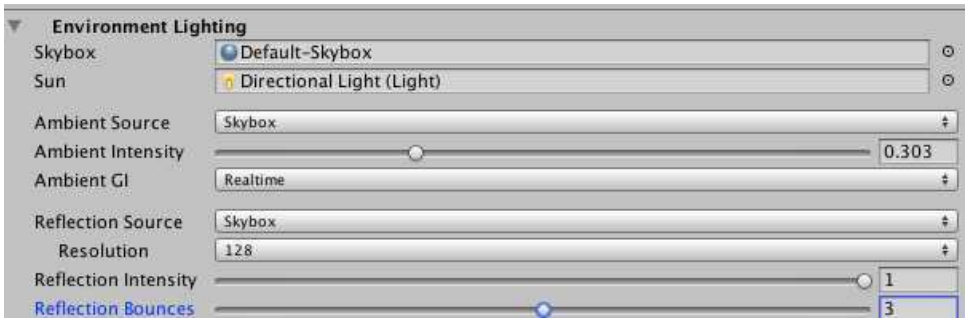
Les objets de la scène sont maintenant prêts pour le calcul des ***lightmaps***.

9. Dans la barre de menus, allez dans :

Window | Lighting

9.1. Dans le panneau *Lighting*, section ***Scene***, changez :

Environment Lighting
9.1.1. ***Sun*** **:** Sélectionnez la lumière **Directional Light.**
9.1.2. ***Ambient Intensity*** : 0.3
9.1.3. ***Reflection Bounces*** : 3



9.2. Appuyez sur « Build » pour calculer les ***lightmaps.***

Après un certain laps de temps, la lumière devrait réapparaître avec ces nouveaux paramètres.

Chaque sphère va apparaître avec une couleur spécifique.

Vous pouvez également désactiver la lumière **Directional light.** L'ambiance demeurera quand même présente.

Notez-bien!

*Il est important de réactiver cette lumière avant de recalculer les **lightmaps.** Sinon, la scène sera totalement noire.*

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu.

L'éclairage restera identique même après avoir relancé une nouvelle partie.

Retournez en mode édition.

La prochaine étape sera de compléter le code relatif au vase. L'ajout d'un point à ***Catch*** quand le personnage touche le vase et un point à ***Miss*** quand celui-ci touche le sol.

Activation du pointage

Ouvrez le script « game.js ».

Ajoutez les lignes suivantes:

```
var GUIGameOver:UI.Text;
var BtnRejouer:GameObject;
private var gameover = false;
var playerObj:GameObject;
private var score = -1;
var GUIScore:UI.Text;
private var miss = 0;
var GUIMiss:UI.Text;

function Start () {
  BtnRejouer.SetActive(false);
  gameover = false;
  GUIGameOver.text = "";
  GUIMiss.text = "MISS: 0";
  GUIScore.text = "CATCH: 0";
}

…
function restart(){
  Application.LoadLevel(Application.loadedLevel);
}

function hitCatch(){
  if (!gameover){
      score++;
      GUIScore.text = "CATCH: "+ score.ToString();
  }
}

function hitGround(){
  if (!gameover){
    miss++;
    GUIMiss.text = "MISS: "+ miss.ToString();
  }
}
```

↘ *Les textes en caractères **gras** sont à ajouter, et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script.

En détails…

Les deux nouvelles fonctions vont incrémenter une variable selon les besoins (**score++** ou **miss++**). Elles vont ensuite afficher le résultat dans les champs textes (**GUIScore.text** et **GUIMiss.text**). Afin de ne pas calculer le premier vase qui tombe durant l'intro, on initialise la ***variable :* score** à -1 (**private var score = -1**). Dans la fonction (**Start()**), on affiche un pointage de 0 au démarrage du jeu.

Ouvrez le script « vase.js ».

Ajoutez le code suivant :

```
var scriptObj:GameObject;
var catchParticles:ParticleSystem;
var missParticles:ParticleSystem;
private var origins:Vector3;

function Start(){
    origins = transform.position;
}

function OnCollisionEnter(col:Collision){
    var objCopy:ParticleSystem;
    var particlesSelect:ParticleSystem;

    if (col.gameObject.tag == "Player"){
      col.gameObject.SendMessage("catchObj");
      scriptObj.SendMessage("hitCatch");
      particlesSelect = catchParticles;
    }
    else if (col.gameObject.tag == "floor"){
       scriptObj.SendMessage("hitGround");
       particlesSelect = missParticles;
    }
    else if (col.gameObject.tag == "junk"){
       replaceObj();
       return;
    }else
       particlesSelect = missParticles;
    objCopy = Instantiate(particlesSelect, transform.position,
                          Quaternion.identity);
    objCopy.Play();
    Destroy(objCopy.gameObject, 2);
    replaceObj();
}
…
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et« … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Quand le vase détecte une collision avec « Player », on appelle cette fonction (**SendMessage("hit-Catch")**). Si la collision se fait avec « floor », c'est la fonction (**SendMessage("hitGround")**) que l'on appelle.

Le code du vase devrait être prêt pour le gameplay. Vous allez assigner quelques variables afin de le compléter.

10. Sélectionnez l'objet **EventSystem** dans *Hierarchy.*

10.1. Dans *l'Inspector*, changez :

Game (Script)
10.1.1. ***GUI Score*** : Glissez le texte **GUI Catch** de votre **CanvasUI.**
10.1.2. ***GUI Miss*** : Glissez le texte **GUI Miss** de votre **CanvasUI.**

11. Sélectionnez l'objet **Vase_Prefab.**

11.1. Dans *l'Inspector*, changez :

Vase (Script)
11.1.1. ***Script Obj*** : glissez l'objet **EventSystem.**

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu.

La jouabilité est maintenant complète. Remarquez que le visage de notre personnage n'est pas très réactif.

Retournez en mode édition.

Ajout d'expressions

Ouvrez le script « character.js ».

Ajoutez les lignes :

```
…
var speed:float = 30;
var lockmouse:boolean = true;
var faceDefault:GameObject;
var faceHappy:GameObject;
var faceSad:GameObject;
var faceStressed:GameObject;
var faceSurprised:GameObject;

function Awake(){
    anim = GetComponent.<Animator>();
    changeFace("");
}
…
function catchObj(){
  anim.SetTrigger("catch");
  if (currentBaseState.IsName("Base Layer.locomotion"))
     changeFace("happy");
  else
     changeFace("surprise");
  yield WaitForSeconds(1);
  changeFace("stress");
}

function sadFace(){
  changeFace("sad");
  yield WaitForSeconds(1);
  changeFace("stress");
}
…
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

En détails…

Quand la fonction catchObj est appelée, on interroge l'Animator Controller. On s'assure que l'on est dans l'intro ou dans la locomotion (**IsName("Base Layer.locomotion")**). Selon le cas, on affiche le visage (**happy** ou **surprise**). Après une seconde (**yield WaitForSeconds(1)**), on retourne au visage (**stress**).

Toujours dans le même script, ajoutez également :

```
…

function changeFace(emotion:String){
   faceDefault.SetActive(false);
   faceSad.SetActive(false);
   faceHappy.SetActive(false);
   faceSurprised.SetActive(false);
   faceStressed.SetActive(false);

   switch (emotion){
      case "sad":
         faceSad.SetActive(true);
      break;
      case "happy":
         faceHappy.SetActive(true);
      break;
      case "surprise":
         faceSurprised.SetActive(true);
      break;
      case "stress":
         faceStressed.SetActive(true);
      break;
      default:
         faceDefault.SetActive(true) ;
      break;
    }
}
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script.

En détails...

Nous avons obtenu la fonction qui change les expressions faciales (**changeFace()**). Au début de l'opération, il faut cacher tous les visages (**SetActive(false)**). Selon la valeur de la variable (**emotion**), on choisit le visage désiré (**SetActive(true)**).

Dans le script « game.js », ajoutez :

```
…
function hitGround(){
   if (!gameover){
      miss++;
      GUIMiss.text = "MISS: "+ miss.ToString();
      playerObj.SendMessage("sadFace");
   }
}
…
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Lorsque le vase touche le sol, vous devez également changer le visage du personnage en appelant la fonction (**sadFace**) dans le script **character.js.**

12. Dans le panneau *Hierarchy,* localisez l'objet **tete** qui se trouve dans **perso.**

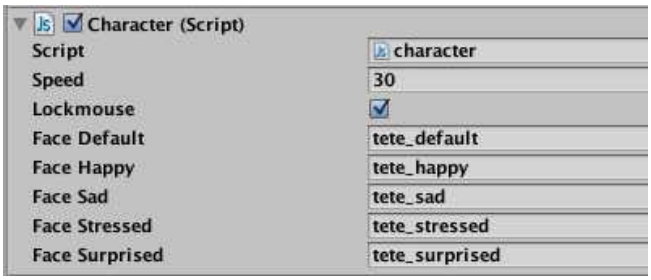
12.1. Localisez les différents visages avec les noms suivants :

12.1.1. tete_default
12.1.2. tete_happy
12.1.3. tete_stressed
12.1.4. tete_sad
12.1.5. tete_surprised

Nous allons y revenir plus loin.

13. Sélectionnez l'objet parent **perso.**

13.1. Dans *l'Inspector*, glissez les visages dans les variables :

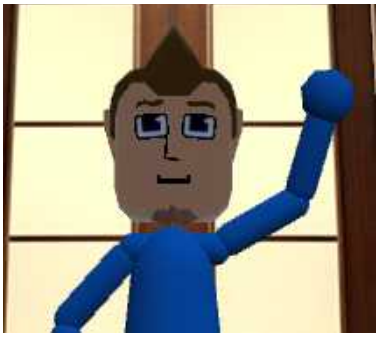


13.1.1. ***Face Default*** **:** glissez **tete_default**
13.1.2. ***Face Happy*** **:** glissez **tete_happy**
13.1.3. ***Face Sad*** **:** glissez **tete_sad**
13.1.4. ***Face Stressed*** **:** glissez **tete_stressed**
13.1.5. ***Face Surprised*** **:** glissez **tete_surprised**

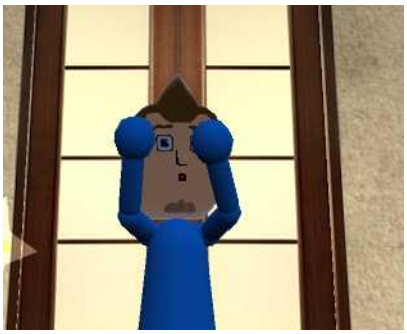
Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu.

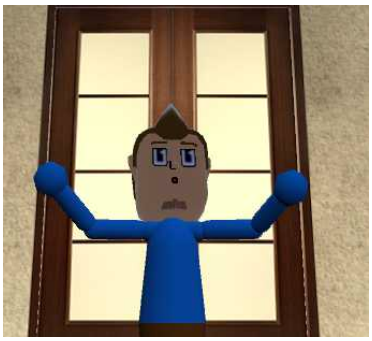
Vous devriez avoir des expressions et animations pour toutes les occasions.



↘ *Visage au début de l'intro*



↘ *Fin de l'intro*



↘ *Par défaut, durant le gameplay*

↘ *Durant l'attrapée*

↘ *Vase manqué*

Retournez en mode édition.

Le jeu est pratiquement complet mais celui-ci est encore muet. Le curseur de la souris est toujours visible lors du déroulement du jeu.

Ajout de sons

Ouvrez le script « game.js », ajoutez :

```
…
var GUIMiss:UI.Text;
var crunchedDeathSFX:AudioClip;
var debutIncendieSFX:AudioClip;
var incendieLoopSFX:AudioClip;
var MCamera:AudioSource;

function Start () {
     Cursor.lockState = CursorLockMode.Locked;
     Cursor.visible = false;
     MCamera.PlayOneShot(debutIncendieSFX);
     Invoke ("PlayLoop", MCamera.clip.length);
     BtnRejouer.SetActive(false);
     gameover = false;
     GUIGameOver.text = "";
     GUIMiss.text = "MISS: 0";
     GUIScore.text = "CATCH: 0";
}

function PlayLoop (){
    MCamera.clip = incendieLoopSFX;
    MCamera.loop = true;
    MCamera.Play();
}

function Update (){
    if (Input.GetButtonUp("Cancel")){
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}
…
function hitPlayer(){
    GUIGameOver.text = "GAME OVER";
    gameover = true;
    playerObj.SetActive(false);
    BtnRejouer.SetActive(true);
    MCamera.PlayOneShot(crunchedDeathSFX);
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;
}
…
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails...

Au démarrage du jeu (**Start()**) il faut verrouiller et cacher le curseur (**Cursor.lockState = CursorLockMode.Locked** et **Cursor.visible**). Le son (debutIncendieSFX) doit être lancé. Lorsqu'il est complété, il faut appeler la fonction **PlayLoop** (**Invoke ("PlayLoop", MCamera.clip.length)**). La fonction (**PlayLoop**) lance le son (**incendieLoopSFX**) en boucle (**MCamera.loop = true**). Quand on appuie sur le bouton (**Cancel**), on débloque et affiche le curseur (**lockState** et **visible**). Finalement, quand le personnage est touché par une boule de feu (**hitPlayer()**), on ajoute le son de la fin du jeu (**PlayOneShot(crunchedDeathSFX)**), on débloque et affiche le curseur (**lockState** et **visible**).

14. Dans *Hierarchy*, sélectionnez **Main Camera.**

14.1. Dans *l'Inspector,* appuyez sur le bouton « Add Component », puis :

Audio | AudioSource

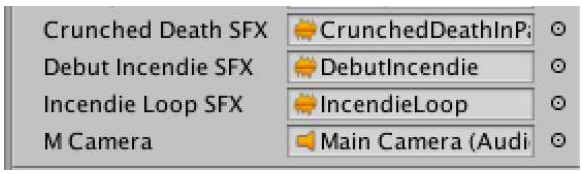
14.2. Dans ***Component***, changez :

Audio Source
14.2.1. ***AudioClip*** : glissez **DebutIncendie.wav** du *Project.*

15. Sélectionnez l'objet **EventSystem.**

15.1. Changez les paramètres suivants :

Game (Script)
15.1.1. ***Crunched Death SFX*** **:** glissez **CrunchedDeathInPain.wav** du *Project.*
15.1.2. ***Debut Incendie SFX*** **:** glissez **DebutIncendie.wav** du *Project.*
15.1.3. ***Incendie Loop SFX*** **:** glissez **IncendieLoop.wav** du *Project.*
15.1.4. ***M Camera*** **:** glissez la **Main Camera** de *Hierarchy.*



Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu.

Lorsque le joueur perd, il devrait y avoir une ambiance sonore de départ et on devrait entendre un son. Le curseur de la souris devrait également être immobilisé durant le déroulement de la partie, à moins d'une défaite ou si on appuie sur la touche (**esc** ⎋) au clavier.

Retournez en mode édition.

Retournez en mode édition.

Vous allez ajouter deux nouvelles sources sonores pour les objets qui tombent du ciel.

Ouvrez le script « junk.js ».

Puis ajoutez :

```
…
var scriptObj:GameObject;
var smokeball:ParticleSystem;
private var AudioS:AudioSource;
var rocksOnFloorSFX:AudioClip;

function Start () {
  origins = transform.position;
  AudioS = GetComponent.<AudioSource>();
}

function OnCollisionEnter(col:Collision){
     if (col.gameObject.tag == "Player")
          scriptObj.SendMessage("hitPlayer");
     if (col.gameObject.tag != "vase"){
        var objCopy = Instantiate(smokeball, transform.position,
                                        Quaternion.identity);
        objCopy.Play();
        AudioS.PlayOneShot(rocksOnFloorSFX);
        Destroy(objCopy.gameObject, 2);
        replaceObj();
     }
}
…
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

En détails…

Lorsqu'une collision est détectée avec le sol, on entend le son (**rocksOnFloorSFX**).

16. Sélectionnez l'objet **junk** dans la scène.

16.1. Dans *l'Inspector,* appuyez sur le bouton « Add Component », puis :

16.2. Changez les paramètres :

Junk (Script)
16.2.1. ***Break On Floor SFX* :** glissez **RocksHitGround.wav** du *Project.*

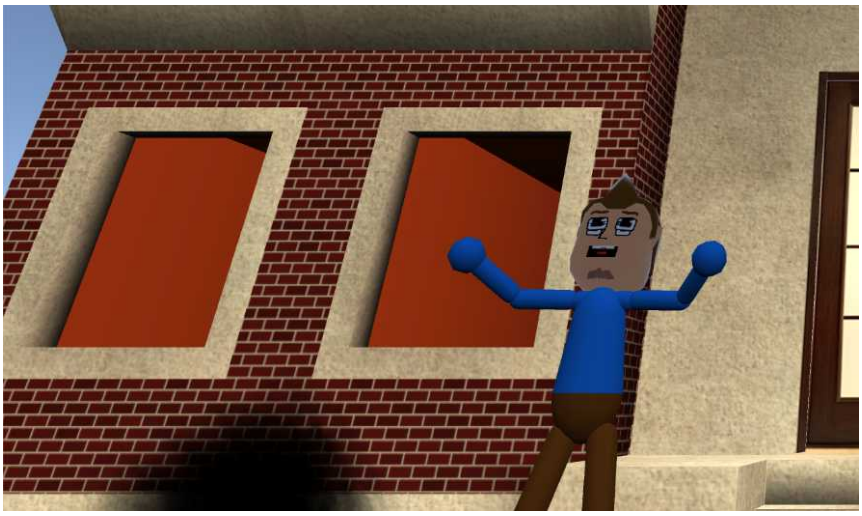
Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu.

Vous devriez entendre un son quand la boule de feu touche le sol.

Retournez en mode édition.

Vous allez exécuter les mêmes fonctions pour le vase afin d'obtenir les mêmes résultats.



Ouvrez le script « vase.js ».

Puis ajoutez :

```
…
private var origins:Vector3;
var glassBreakSFX:AudioClip;
var collectItemSFX:AudioClip;
private var AudioS:AudioSource;

function Start(){
    private var origins:Vector3;
    AudioS = GetComponent.<AudioSource>();
}

function OnCollisionEnter(col:Collision){
    var objCopy:ParticleSystem;
    var particlesSelect:ParticleSystem;

    if (col.gameObject.tag == "Player"){
        col.gameObject.SendMessage("catchObj");
        scriptObj.SendMessage("hitCatch");
        particlesSelect = catchParticles;
        AudioS.PlayOneShot(collectItemSFX);
    }else if (col.gameObject.tag == "floor"){
        scriptObj.SendMessage("hitGround");
        particlesSelect = missParticles;
        AudioS.PlayOneShot(glassBreakSFX);
    }else if (col.gameObject.tag == "junk"){
        replaceObj();
        return;
    }else
        particlesSelect = missParticles;
    objCopy = Instantiate(particlesSelect, transform.position,
                          Quaternion.identity);
    objCopy.Play();
    Destroy(objCopy.gameObject, 2);
    replaceObj();
}
…
```

↘ *Les textes en **gras** sont à ajouter, le reste demeure inchangé et « … » est utilisé pour simplifier la lecture.*

Sauvegardez le script et retournez dans l'éditeur.

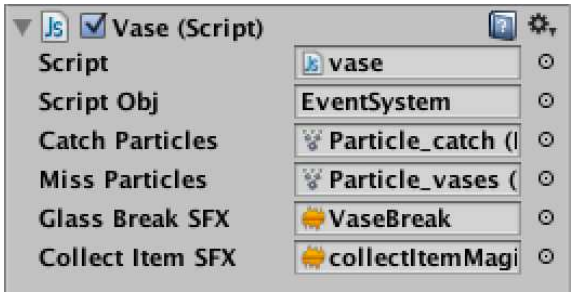
En détails...

Quand une collision est détectée avec le sol, on entend le son (**glassBreakSFX**). Quand la collision est détectée avec le personnage, on entend le son (**collectItemSFX**).

17. Sélectionnez l'objet **Vase_Prefab** dans *Hierarchy*.

17.1. Dans *l'Inspector,* appuyez sur le bouton « Add Component », puis :

Audio | AudioSource

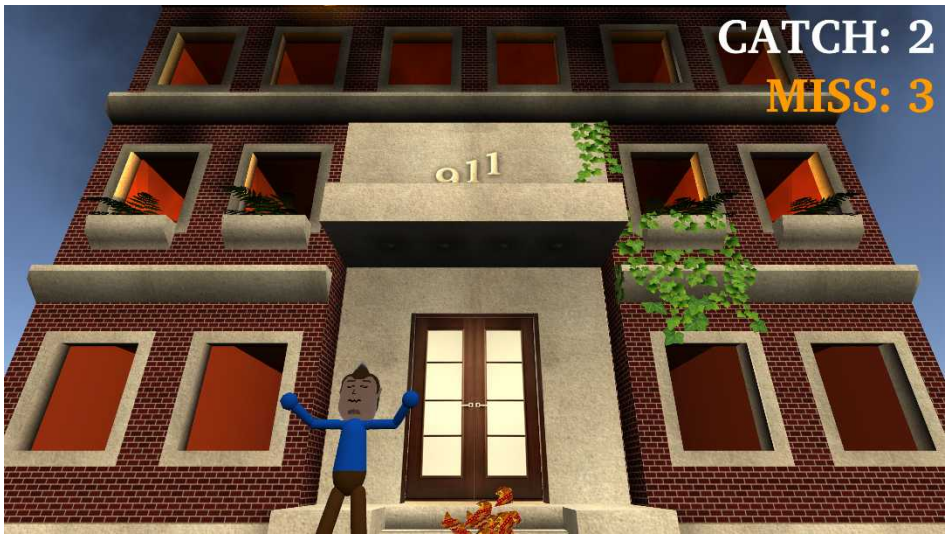


17.2. Changez les paramètres :

17.2.1. ***Glass Break SFX*** **:** VaseBreak.wav
17.2.2. ***Collect Item SFX*** **:** collectItemMagic.wav

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

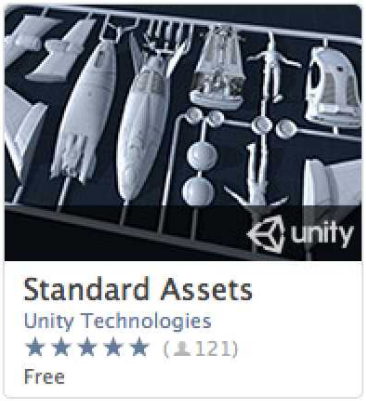
Testez le jeu.



Retournez en mode édition.

Le jeu est techniquement complet, à l'exception d'une ambiance visuelle.

Ajout d'une ambiance visuelle



Pour cette étape, nous utiliserons le package **Standard Assets** qui est disponible à partir de *Asset Store.*

18. À partir de la barre de menus, allez :

Window | Asset Store

19. À partir du magasin, recherchez **Standard Assets.**

20. Téléchargez (***Download***) ou importez (***Import***) le package.

Notez-bien!

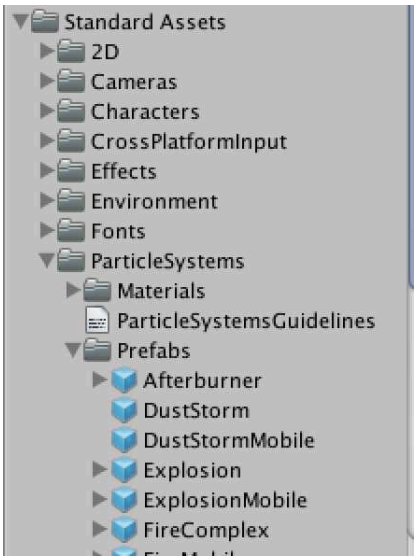
Faites attention de bien télécharger la version pour Unity 5 et non celle pour Unity 4.6.

21. Importez le contenu du ***package*** dans votre projet.

Premièrement, vous allez ajouter des flammes dans les fenêtres.

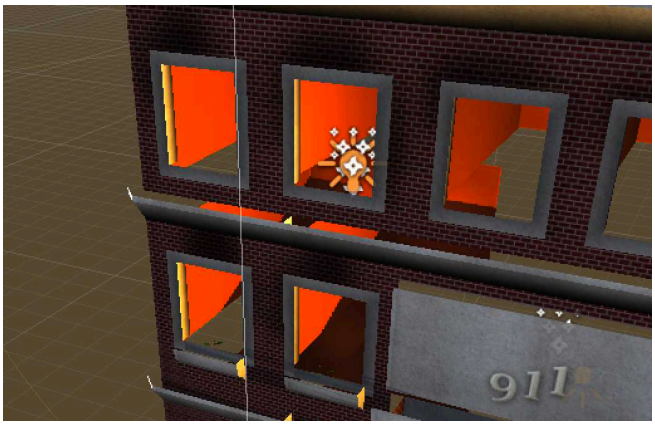
22. Dans le panneau *Project,* allez dans répertoire :

Standard Assets | ParticleSystems | Prefabs



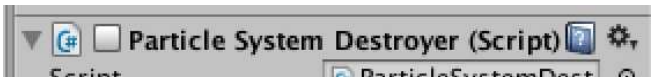
23. Glissez le ***prefab*** **FireComplex** dans la scène.

23.1. Positionnez-le dans une des fenêtres.



23.2. Sur cet objet dans *Inspector*,

23.2.1. Décochez le script ***Particle System Destroyer.***

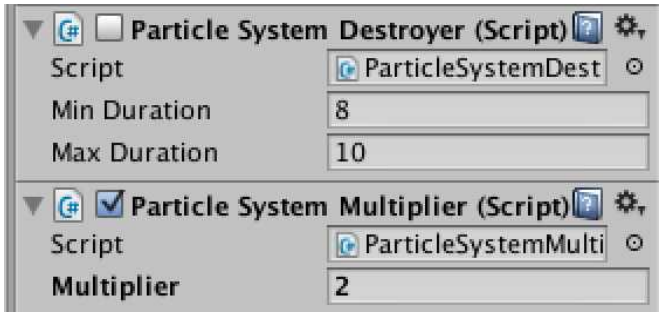


En détails...

Il faut désactiver ce ***script***, car le feu doit persister durant tout le ***gameplay.***

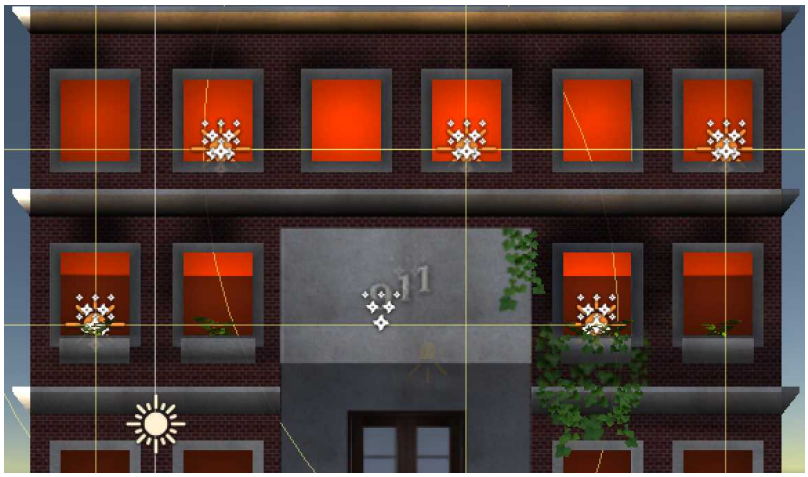
23.3. Dans l'Inspector, changez le paramètre suivant :

Particle System Multiplier
23.3.1. ***Multiplier*** : 2



24. Faites quelques copies du ***prefab*** de la scène avec (**CTRL+D**) ou (**⌘+D**) sur Mac.

24.1. Placez les copies dans différentes fenêtres (tentez de ne pas surcharger la scène).



Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu.

Il devrait y avoir des flammes qui sortent des fenêtres.

Retournez en mode édition.

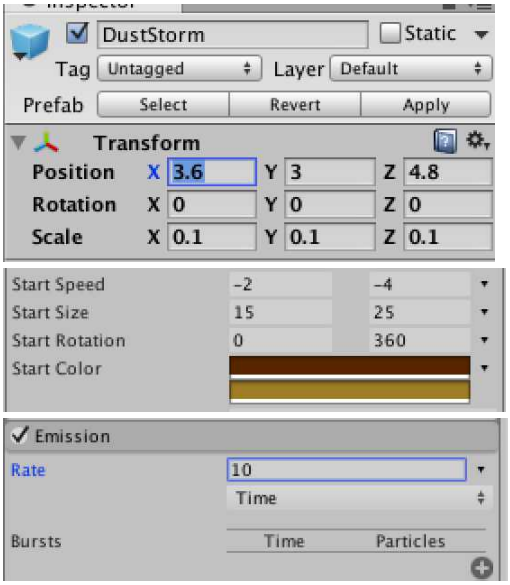
Vous allez ajouter une fumée dense autour des lieux.

25. Dans le panneau *Project,* allez dans le répertoire :

Standard Assets | ParticleSystems | Prefabs

26. Glissez le ***prefab*** **DustStorm** dans la scène.

26.1. Positionnez le ***prefab*** au milieu de la scène



26.2. Dans l'Inspector, changez :

Transform
26.2.1. ***Scale*** **X=** 0.1**, Y=** 0.1**, Z =** 0.1

26.3. Dans le ***Particle System,*** changez :

Paramètres de base
26.3.1. ***Start Speed*** **:** -2 et -4
26.3.2. ***Start Color*** **:** Rouge et Orange

Emission
26.3.3. ***Rate*** **=** 10

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu.

Une fumée dense devrait être présente dans la scène.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.

Testez le jeu.

Voilà, vous avez maintenant complété l'exercice.

Retournez en mode édition.

Codes complétés

Voici les codes complets non abrégés de l'exercice:

character

```
#pragma strict

static var currentBaseState:AnimatorStateInfo;
private var anim:Animator;
var speed:float = 30;
var lockmouse:boolean = true;
var faceDefault:GameObject;
var faceHappy:GameObject;
var faceSad:GameObject;
var faceStressed:GameObject;
var faceSurprised:GameObject;

function Awake(){
  anim = GetComponent.<Animator>();
  changeFace("");
}

function Update () {
  currentBaseState = anim.GetCurrentAnimatorStateInfo(0);
  if (!lockmouse){
     var trans_X = Input.GetAxis("Mouse X") * speed;
     trans_X = trans_X * Time.deltaTime;
     anim.SetFloat("horizontal", Input.GetAxis("Mouse X"));
     transform.Translate (-trans_X, 0,0);
     if (transform.position.x > 7.5){
        transform.position.x = 7.5;
     }
     if (transform.position.x < 0){
        transform.position.x = 0;
     }
  }else if (currentBaseState.IsName("Base Layer.locomotion")){
     lockmouse = false;
  }
}
```

character (fin)

```
function catchObj(){
  anim.SetTrigger("catch");
  if (currentBaseState.IsName("Base Layer.locomotion"))
       changeFace("happy");
  else
       changeFace("surprise");
  yield WaitForSeconds(1);
  changeFace("stress");
}

function sadFace(){
  changeFace("sad");
  yield WaitForSeconds(1);
  changeFace("stress");
}

function changeFace(emotion:String){
  faceDefault.SetActive(false);
  faceSad.SetActive(false);
  faceHappy.SetActive(false);
  faceSurprised.SetActive(false);
  faceStressed.SetActive(false);

  switch (emotion){
     case "sad":
        faceSad.SetActive(true);
     break;
     case "happy":
        faceHappy.SetActive(true);
     break;
     case "surprise":
        faceSurprised.SetActive(true);
     break;
     case "stress":
        faceStressed.SetActive(true);
     break;
     default:
        faceDefault.SetActive(true) ;
     break;
  }
}
```

game

```
#pragma strict

var GUIGameOver:UI.Text;
var BtnRejouer:GameObject;
private var gameover = false;
var playerObj:GameObject;
private var score = -1;
var GUIScore:UI.Text;
private var miss = 0;
var GUIMiss:UI.Text;
var crunchedDeathSFX:AudioClip;
var debutIncendieSFX:AudioClip;
var incendieLoopSFX:AudioClip;
var MCamera:AudioSource;

function Start () {
  Cursor.lockState = CursorLockMode.Locked;
  Cursor.visible = false;
  MCamera.PlayOneShot(debutIncendieSFX);
  Invoke ("PlayLoop", MCamera.clip.length);
  BtnRejouer.SetActive(false);
  gameover = false;
  GUIGameOver.text = "";
  GUIMiss.text = "MISS: 0";
  GUIScore.text = "CATCH: 0";
}

function PlayLoop (){
  MCamera.clip = incendieLoopSFX;
  MCamera.loop = true;
  MCamera.Play();
}

function Update (){
  if (Input.GetButtonUp("Cancel")){
     Cursor.lockState = CursorLockMode.None;
     Cursor.visible = true;
  }
}
```

game (fin)

```
function hitPlayer(){
  GUIGameOver.text = "GAME OVER";
  gameover = true;
  playerObj.SetActive(false);
  BtnRejouer.SetActive(true);
  MCamera.PlayOneShot(crunchedDeathSFX);
  Screen.lockCursor = false;
}

function restart(){
  Application.LoadLevel(Application.loadedLevel);
}

function hitCatch(){
  if (!gameover){
      score++;
      GUIScore.text = "CATCH: "+ score.ToString();
  }
}

function hitGround(){
  if (!gameover){
    miss++;
    GUIMiss.text = "MISS: "+ miss.ToString();
    playerObj.SendMessage("sadFace");
  }
}
```

junk

```
#pragma strict

private var origins:Vector3;
var scriptObj:GameObject;
var smokeball:ParticleSystem;
private var AudioS:AudioSource;
var rocksOnFloorSFX:AudioClip;

function Start () {
  origins = transform.position;
  AudioS = GetComponent.<AudioSource>();
}

function OnCollisionEnter(col:Collision){
  if (col.gameObject.tag == "Player")
     scriptObj.SendMessage("hitPlayer");

  if (col.gameObject.tag != "vase"){
     var objCopy = Instantiate(smokeball, transform.position,
                                        Quaternion.identity);
     objCopy.Play();
     AudioS.PlayOneShot(rocksOnFloorSFX);
     Destroy(objCopy.gameObject, 2);
     replaceObj();
  }
}

function replaceObj(){
  transform.rotation = Quaternion.Euler(Vector3.zero);
  origins.x = Random.Range(0.5, 7.0) ;
  transform.position = origins;
  GetComponent.<Rigidbody>().velocity = Vector3.zero;
}
```

vase

```
#pragma strict

var scriptObj:GameObject;
var catchParticles:ParticleSystem;
var missParticles:ParticleSystem;
private var origins:Vector3;
var glassBreakSFX:AudioClip;
var collectItemSFX:AudioClip;
private var AudioS:AudioSource;


function Start(){
  origins = transform.position;
  AudioS = GetComponent.<AudioSource>();
}

function OnCollisionEnter(col:Collision){
  var objCopy:ParticleSystem;
  var particlesSelect:ParticleSystem;

  if (col.gameObject.tag == "Player"){
     col.gameObject.SendMessage("catchObj");
     scriptObj.SendMessage("hitCatch");
     particlesSelect = catchParticles;
     AudioS.PlayOneShot(collectItemSFX);
  }else if (col.gameObject.tag == "floor"){
     scriptObj.SendMessage("hitGround");
     particlesSelect = missParticles;
     AudioS.PlayOneShot(glassBreakSFX);
  }else if (col.gameObject.tag == "junk"){
     replaceObj();
     return;
  }else
     particlesSelect = missParticles;
  objCopy = Instantiate(particlesSelect, transform.position,
                        Quaternion.identity);
  objCopy.Play();
  Destroy(objCopy.gameObject, 2);
  replaceObj();
}

function replaceObj(){
  transform.rotation = Quaternion.Euler(Vector3.zero);
  origins.x = Random.Range(0.5, 7.0) ;
  transform.position = origins;
  GetComponent.<Rigidbody>().velocity = Vector3.zero;
}
```

Chapitre 12 - Le développement mobile

Matière couverte dans ce chapitre

- **Les origines**
- **Ce qu'il faut savoir**
- **Comment se démarquer de la compétition**
- **Le *Freemium***
- **Comment concevoir une interface utilisateur mobile**
- **Comment configurer et utiliser Unity Remote**
- **Comment compiler et exporter pour Android**
- **Comment compiler et exporter pour iOS**

Les téléphones portables ont gagné en popularité depuis le début du millénaire. En 2013, plus de 1.8 milliard d'appareils se sont vendus dans le monde[1]! Près de 1 milliard de ces appareils sont des téléphones intelligents. Ce n'est pas une nouveauté car en 2006 on frôlait déjà 1 milliard de téléphones portables vendus annuellement dans le monde.

Avec autant d'utilisateurs, il est indéniable que les appareils mobiles représentent une immense opportunité d'affaire pour les développeurs. Cependant, avec plus d'un million d'applications qui se font compétition pour une part du marché, il est important de savoir se démarquer!

Il faut également connaître comment se font: le développement d'un jeu mobile, le démarrage, le fonctionnement pour ce type d'appareils et la publication du jeu sur les différentes plateformes disponibles. Ce chapitre a pour objectif de répondre à ces interrogations.

Cette théorie sera séparée en **trois sections** :

• **Les origines:** où l'on prendra connaissance de l'évolution des appareils et des systèmes d'exploitation qu'ils utilisent.
• **Ce qu'il faut savoir:** comment l'on traite des stratégies de mise en marché et de monétisation pour vos apps.
• **La partie pratique :** cette section abordera concrètement tout ce que vous devriez connaître pour créer un jeu mobile avec Unity®

Les origines

Les jeux mobiles ont fait leur apparition en 1997 avec le classique « **Snake** » chez *Nokia* avec la série *61XX*. Le premier jeu était intégré sur tous les appareils et aucune option n'était disponible pour en installer d'autres.

Les utilisateurs ont dû patienter jusqu'en 2000 avant que d'autres compagnies de jeux fassent leur apparition grâce au protocole *WAP*. Ceux-ci utilisaient des écrans monochromes avec des résolutions de moins de 8000 pixels (total); comme exemple les premiers ordinateurs des années 70 et 80.

L'ajout de *Java ME* dans les téléphones portables à partir de 2001 a permis à des développeurs professionnels de créer des jeux dans un langage connu. Il fallait les distribuer grâce à des ententes avec les fournisseurs de services. Un problème que les développeurs de l'époque devaient affronter était les faibles redevances payées par ces fournisseurs. Dans certains cas, ce chiffre pouvait représenter moins de 50% du prix de vente. De plus, afin d'avoir le droit de publier un titre, les développeurs devait tester et adapter leurs jeux avec la majorité des appareils vendus par le fournisseur. Ce processus d'adaptation et de contrôle de la qualité pouvait prendre entre 9 et 15 mois et demandait de redésigner certains jeux trop complexes pour les appareils bas de gamme. Le tout était très coûteux

[1] *Source : Gartner (Février 2014)*

pour les développeurs et les *publishers* qui les subventionnent. Sans compter que les *publishers* prenaient une large part du marché (entre 50 et 80% des bénéfices) et qu'en plus, 95% des jeux étaient téléchargés illicitement. Malgré ces contraintes, les jeux mobiles connurent un énorme succès international grâce à une montée phénoménale de la demande pour les téléphones.

En 2003, la compagnie Finlandaise *Nokia* voyait une opportunité de combiner l'expérience d'une console de jeux portable ainsi que des fonctionnalités d'un lecteur MP3 dans un téléphone mobile nommé *N-Gage*. Cette première tentative afin de combiner les différents médiums ne fut pas un succès; notamment dû à un mauvais design et à un prix d'achat élevé. Parmi les problèmes les plus flagrants, il y avait que le compartiment pour insérer le jeu se trouvait derrière la batterie, ce qui demandait à l'usager d'éteindre son portable avant de changer de jeu. L'écran était plus haut que long, ce qui limitait beaucoup la vision latérale des jeux. De plus, dû à un design excentrique, le haut parleur et le microphone du téléphone se trouvaient sur la frange de l'appareil, forçant l'usager à tenir le téléphone sur le côté pour prendre un appel. On aurait dit que les usagers tenaient un tacos contre leur oreille. Plusieurs mauvaises blagues sur le net ont alors apparu.

C'est à cette même époque que *RIM* (maintenant *BlackBerry Limited*) lança ses premiers téléphones intelligents : le *BlackBerry RIM 850* et *857*. Ces appareils, en plus de pouvoir envoyer et recevoir des appels, permettaient aux usagers de recevoir des emails, d'échanger des messages textes, d'envoyer des télécopieurs via Internet, de naviguer sur le net et beaucoup plus. *RIM* a connu à cette époque une montée rapide de la demande, au point où l'on a surnommé ce phénomène *CrackBerry* dû à la dépendance que cet appareil a créé chez les utilisateurs.

En 2006, on estimait que les ventes de téléphones portables allaient frôler le cap du milliard d'appareils vendus annuellement et que la demande continuerait à grimper. C'est d'ailleurs la raison donnée en 2007 par Steve Jobs, le créateur d'Apple, pour justifier que sa compagnie se lance dans le marché des téléphones mobiles. À l'époque, *Apple* était réticent à l'idée que d'autres développeurs puissent faire des applications pour leur précieux gadget. Durant toute une année, le *iPhone* avait l'exclusivité du marché.

Ce n'est qu'avec la sortie du *iPhone 3G* en 2008, qu'*Apple* introduisit son kit de développement officiel pour faire des applications (ou *apps*). Ils sont distribués exclusivement par *Apple* pour tous les téléphones *iPhone* existants.

L'*Apple App Store* introduisit de grands changements dans l'industrie du jeu mobile, il changera pour toujours le développement de jeux portables :

• Désormais, les développeurs peuvent publier eux-mêmes leurs jeux, sans passer par un *publisher* ou un fournisseur de service. *Apple* se réserve la distribution exclusive des applications sur leurs appareils. Quelques privilèges sont toutefois accordés pour les entreprises.

• Les développeurs indépendants jouissent des mêmes privilèges et de la même visibilité que les grosses compagnies.

• Les *apps* sont téléchargées directement en ligne sur le site d'*Apple.* Selon *Apple*, 90% des apps soumises sont approuvées en moins de 10 jours, ce qui réduit de beaucoup le temps de négociation et de distribution des jeux.

• Finalement, *Apple* réclame seulement 30% de commission sur les applications vendues. Le montant qui reste va directement dans les poches du développeur.

Malgré les limitations techniques et le contrôle sur le contenu exercé par *Apple*, *l'Apps Store* est un succès immédiat. Plusieurs histoires de réussites émergent, créant un engouement chez les développeurs qui espèrent toucher le gros lot!

Évidemment, ce n'était qu'une question de temps avant qu'un compétiteur vienne reproduire le modèle d'*Apple*. En 2009, *Google* fait une annonce qui aura l'effet d'une bombe sur *Apple*; il se lance sur le marché des téléphones intelligents avec *Android*, un système d'exploitation qu'il vient d'acquérir. La controverse vient du fait qu'Eric Schmidt (CEO de *Google* à l'époque), faisait partie du conseil d'administration d'*Apple* au moment de la création de l'*iPhone* et qu'*Android* reproduisait une expérience similaire au système d'exploitation *iPhone OS* (maintenant nommé *iOS*).

Avec l'arrivée d'un nouveau joueur sur le marché des téléphones intelligents, un problème survient : la fragmentation du marché! D'un côté, *Apple* contrôle soigneusement son écosystème. Les *apps* doivent être approuvées et vendues par *Apple*, qui se réserve le droit de bannir certains développeurs si ceux-ci violent les règles mises en place. Les apps sont uniquement compatibles avec les appareils vendus par *Apple*. D'un autre côté, *Google* distribue ouvertement son système *Android*, ce qui veut dire que n'importe quel fabriquant de téléphones portables peut utiliser ou modifier une version *d'Android* pour leurs appareils. Les *apps* peuvent être obtenues via plus d'une centaine de différents distributeurs et *Google* est plus libéral avec les soumissions faites sur *Google Play Store*.

Il a fallu attendre jusqu'en 2011 pour que *Microsoft* se lance sérieusement sur le marché mobile avec *Windows 7 phone* et plus rigoureusement en 2012 avec *Windows 8 phone*. En plus d'unifier leur architecture (passant du *noyau CE à NT* comme sur PC), les gestionnaires ont également adopté une stratégie de distribution similaire à *Apple*, où ils contrôlent l'approbation des applications qui se retrouvent dans leur catalogue. Cependant, comme ils sont arrivés en dernier sur le marché, ils doivent accepter plusieurs programmes de qualités médiocres dans leur inventaire. Pour ajouter à leur malheur, *Microsoft* est boudé par les manufacturiers les plus importants. C'est pourquoi en 2011, *Microsoft* s'est allié avec *Nokia* pour remplacer leur système existant *Symbian* pour *Windows phone 7*. Puis en 2013, *Microsoft* décide d'acheter *Nokia* afin de pousser *Windows* auprès des usagers de la marque Finlandaise.

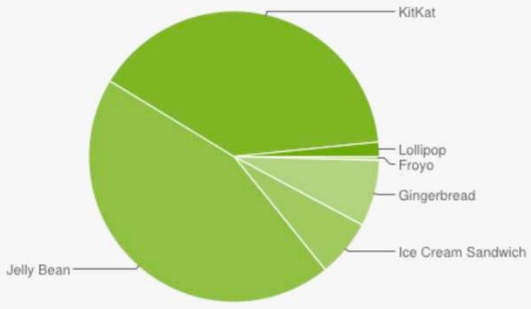
Depuis 2011, *Android* est le système d'exploitation mobile le plus populaire, surpassant *Apple* et *BlackBerry*.

Ce qu'il faut savoir

En 2014, *Android* se retrouve dans plus de **47%** des appareils actifs. *Apple iOS* dans plus de **42%** et *Windows phone* compte pour un peu plus de **2%**. Les autres systèmes utilisés incluent : le système discontinué *Symbian* avec **3%** et l'ancien leader mobile *BlackBerry-10* avec moins de **1%**.

En regardant les chiffres qui précèdent, on pourrait conclure qu'*Android* est la plateforme de développement idéal. Si on analyse en détails la fragmentation de cette plateforme uniquement, on remarque que la fragmentation rend le développement plus compliqué.

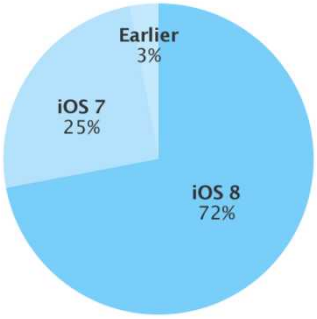
Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	7.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	6.4%
4.1.x	Jelly Bean	16	18.4%
4.2.x		17	19.8%
4.3		18	6.3%
4.4	KitKat	19	39.7%
5.0	Lollipop	21	1.6%



↘ *Données officielles de Google collectées sur une période de 7 jours, se terminant le 2 Février 2015.*

Le principal problème auquel *Google* fait face, c'est qu'il dépend de la collaboration des fabricants d'appareils mobiles pour adapter, tester et déployer les mises à jour d'*Android* sur les différents modèles. Malheureusement, plusieurs compagnies ne font pas de supports de logiciels à long terme pour leurs produits.

En comparaison, *Apple iOS* déploie ses mises à jours sur tous leurs appareils supportés. Les usagers adoptent rapidement les nouvelles versions d'*iOS*, ce qui réduit beaucoup la fragmentation de cette plateforme.



↘ *Données officielles d'Apple collectées le 2 Février 2015.*

La fragmentation du marché est toujours à considérer lorsqu'on publie une application. Supporter uniquement la dernière version du système d'exploitation, peut vous coûter des ventes et insulter les

usagers qui ne peuvent pas mettre leur système à jour. Mais il y a un problème encore plus grand que la fragmentation du marché, celui de la compétition féroce.

Comment se démarquer de la compétition

Avant de mentionner la façon de se démarquer, faisons un exercice de math ensemble.

• En 2010, il y avait 225,000 applications sur le *Apple App Store*.

• 74% de ces programmes étaient payants.

• Au même moment, *Apple* annonçait que $1,000,000,000 US avait été versé aux développeurs.

• Combien en moyenne les développeurs obtiennent par application payante?

La réponse : $6,006 US par application

Le coût de développement moyen est de $35,000 US par application. Mais ce n'est qu'une moyenne, la réalité est que très peu d'applications est rentable et seulement une poignée de développeurs réussissent à faire de gros profits.

Comment peut-on alors maximiser nos chances de faire du profit?

1. **En invitant les joueurs à évaluer positivement votre application.**

↘ *Infinity Blade est une franchise de Chair Entertainment Group, LLC.*

En détails…

Cette approche requiert que le joueur joue au jeu suffisamment longtemps pour donner une appréciation de son expérience. Quand cette condition est rencontrée, alors le jeu affiche un message adressé directement au joueur, l'invitant à écrire ses commentaires en ligne. Quand il indique son intérêt, un lien est ouvert sur la page d'évaluation.

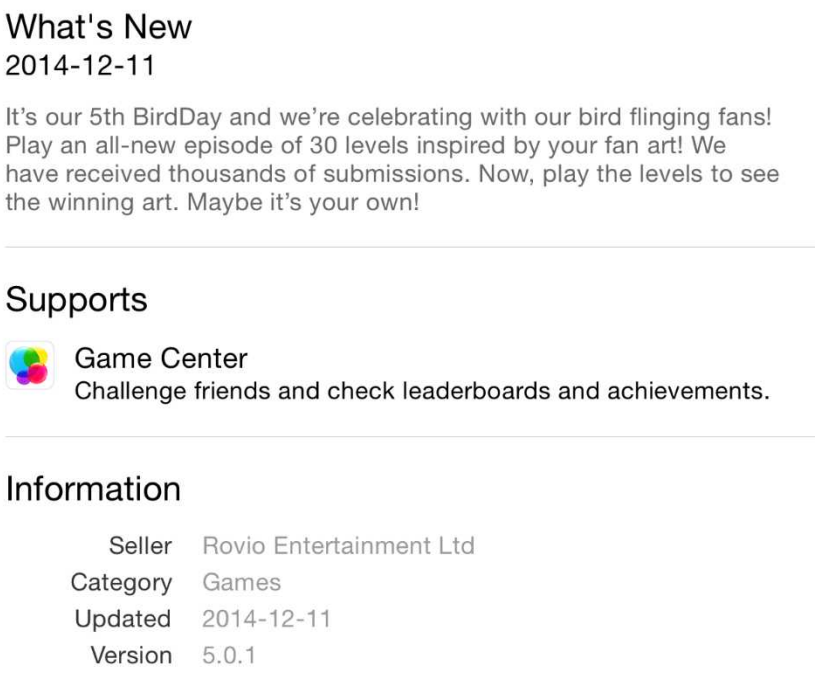
Les avantages :

- Permet de gagner de la popularité, ce qui influence le classement de l'application.
- Incite d'autres joueurs à expérimenter le produit.

Le risque :

- Le joueur irrité peut décider de donner une mauvaise note au jeu.

2. **Avec des mises à jour régulières**



↘ *Angry Bird par Rovio Entertainment Ltd s'est rendu à la version 5.0.1 depuis sa sortie en 2009.*

En détails...

Les mises à jour permettent de réinitialiser les évaluations précédentes, conservant ainsi un historique des différentes évaluations au fil du temps.

Les avantages :

• Permet aux évaluateurs de réévaluer le produit une seconde fois et permet aussi de relancer la popularité d'une ancienne application.
• Les applications qui sont entretenues ont un meilleur cycle de vie que les autres.

Le risque :

• Un joueur qui n'apprécie pas le changement peut critiquer négativement et peut faire chuter la popularité de l'application.

3. **Avec des chutes de prix durant les jours de fêtes**

En détails...

Savoir diminuer le prix permet de gagner de la popularité surtout pendant la période des fêtes. Quand celle-ci est terminée, le prix revient à son prix original. La compagnie peut alors jouir de ventes additionnelles et d'une bonne rentabilité durant les périodes d'affluence.

Les avantages :

- Permet de gagner en popularité et d'augmenter le volume des ventes.
- Génère des ventes additionnelles lorsque le prix revient au prix courant.

Le risque :

- Un joueur qui n'apprécie pas le changement trop rapide de prix peut critiquer cette façon de faire risquant alors de faire chuter la popularité de l'application.

4. **Finalement, avec du portage**

En détails...

Idéalement, les débuts doivent se faire avec une première version sur *iOS* ou *Android* pour tester la demande du marché. Ensuite, on pourra transporter le jeu sur d'autres plateformes et appareils.

L'avantage :

• Permet d'augmenter la clientèle.

Le risque :

• Demande plus de temps, de support et plus d'investissements pour le portage.

Le Freemium (Free2Play)

Le *Freemium* est un nouveau modèle économique où le jeu est distribué gratuitement (à priori) mais incorporant des micro-transactions afin de générer des profits. À première vue, ce modèle peut sembler moins intéressant, mais il est 5 fois plus rentable que le modèle de vente traditionnel!

Ce modèle de vente est basé sur l'additivité du jeu et il ressemble étrangement à la vente de d'autres produits addictifs illicites!

La définition

Le terme *Freemium* vient des mots anglais : «FREE» (gratuit) et «PREMIUM» (prémium).

Comment ce modèle fonctionne

Tout d'abord, on attire l'usager en lui proposant un jeu **gratuitement**. Ensuite, on lui donne un avant-goût du jeu avec une session suffisamment longue pour que le joueur adopte le jeu.

Lorsqu'on s'aperçoit que le joueur aime le jeu, on lui offre des micro-transactions qui vont lui permettre de jouer encore plus longtemps. À partir de ce moment, on demande de plus en plus d'argent durant sa progression. Même s'il se rend compte qu'il a beaucoup investi dans le jeu, le joueur veut continuer à jouer.

En peu de temps, le joueur devient accroc au jeu et il investit de plus en plus. Ce modèle économique rapporte 5 fois plus que les jeux vendus traditionnellement.

Il est possible de faire plus d'argent avec des jeux *Freemium*, car le nombre d'utilisateurs est beaucoup plus élevé qu'un jeu traditionnellement payant. C'est donc sur le volume de ventes que les profits sont réalisés. Comme le jeu est gratuit, le nombre de versions piratées du jeu est beaucoup plus faible, car il requiert d'être en constante communication avec le serveur des ventes qui filtre les applications illégales.

Il existe également un dérivé de ce modèle de vente qui se nomme le *Paymium* et qui consiste à faire payer le joueur pour une expérience complète de jeu, mais on y ajoute des micro-transactions pour tricher ou accélérer la progression du jeu. Cette alternative a déjà meilleure réputation que le *Freemium*, mais elle reste moins rentable que celui-ci.

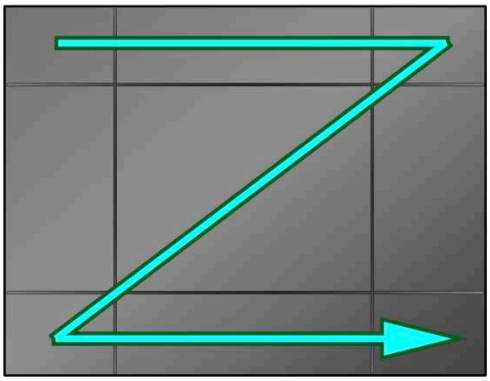
Différentes tactiques de ventes ont été décrites, maintenant nous allons découvrir des interfaces mobiles.

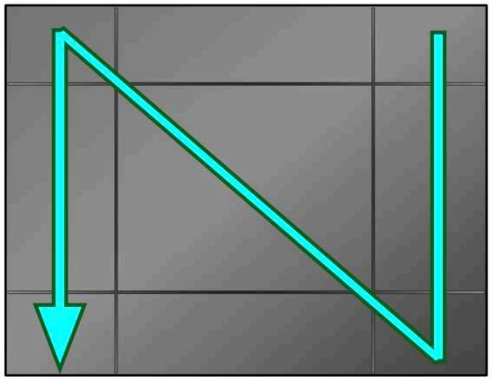
Comment concevoir une interface utilisateur

Le positionnement des éléments à l'écran a un impact sur la perception que l'on se fait d'eux. Certains endroits sont plus faciles d'accès et d'autres tendent à être plus visibles. Le graphique suivant montre comment la visibilité et l'accessibilité sont réparties à l'écran. Le coin supérieur gauche est le plus important dans les civilisations occidentales et le moins important se trouve dans le coin en bas à droite. Pensez à la façon dont vous lisez un livre. Vous débutez une nouvelle page à partir du coin supérieur gauche et vous terminez en bas à droite! Quand vous lisez une page Web, vous répétez sans le savoir les mêmes patterns de lecture. Le centre de l'écran est la zone la plus dominante et la plus accessible. C'est donc à cet endroit que le jeu se déroule.

Notez-bien!

Dans les sociétés orientales, la lecture s'effectue différemment. Il est important de tenir compte des différences culturelles quand on crée une interface pour une clientèle internationale !





↘ *Les cultures occidentales lisent de gauche à droite et de haut vers le bas. Dans les cultures orientales, les gens lisent de haut vers le bas et de droite vers la gauche.*

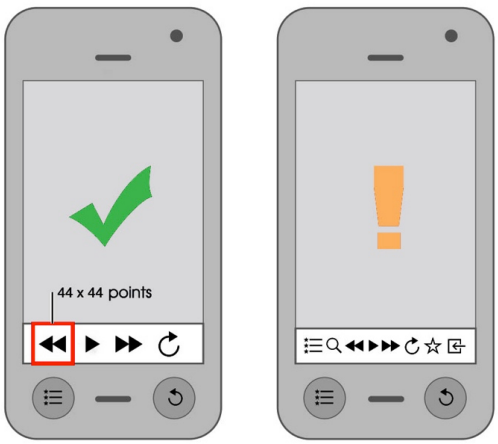
Évitez de surcharger vos interfaces

Conseil!

Quand vous créez vos interfaces, il est important de se poser la question : Est-ce que cet élément est essentiel pour les besoins de l'usager ?

L'erreur la plus souvent commise est de vouloir rendre tous les outils accessibles simultanément dans une seule interface.

Il est préférable de trouver le moyen le plus efficace de regrouper et de disposer les éléments.



Conseil!

Comme disait Steve Jobs aux dirigeants d'Apple en parlant de la philosophie du design de leurs produits « Less is more » (Moins c'est plus). Ce qui veut dire que dans certains cas, retirer les éléments non essentiels peut s'avérer un plus.

Les interfaces sont moins intimidantes pour les novices lorsque les paramètres les plus souvent employés sont prédéfinis à l'avance.

L'image suivante illustre deux interfaces pour un lecteur de musique. L'interface de droite contient un éventail de fonctions dont la plupart n'est rarement utilisée. L'image de gauche montre une interface épurée avec des boutons accessibles qui ont une taille respectable pour un écran de téléphone portable.

Maintenant que nous avons vu les bases générales, nous allons aborder le sujet du développement sur les plateformes mobiles.

La partie pratique

Unity Remote

Lorsque vous développez un jeu pour Android ou iOS (iPhone et iPad), il est difficile de tester les interactions avec les différents senseurs ou avec l'écran tactile sur un PC. Celui-ci ne possède pas ces technologies. Par conséquent, le meilleur moyen de tester un jeu mobile, c'est de le faire directement sur un appareil portable ou une tablette.

Un second problème se présente alors : le temps pour compiler et déployer une version réduit la productivité. Balancer la jouabilité peut devenir un cauchemar.

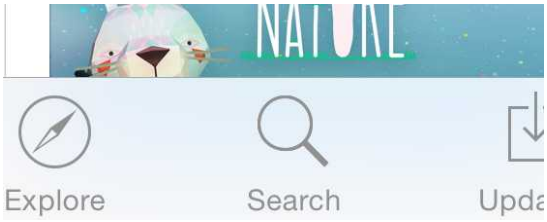
Voici maintenant la solution à ces problèmes : **Unity Remote4** pour Android et iOS; une courte application est aussi disponible sur **Google Play Store** et **Apple App Store**.

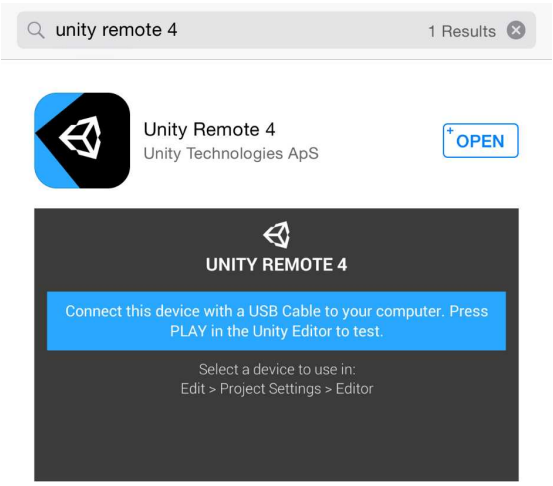
Comment configurer Unity Remote

Installation sur iOS

Atelier pratique

1.À partir de votre iPhone ou iPad, ouvrez votre **App Store**.





2. À partir de **l'App Store**, appuyez sur le bouton « recherche » dans le bas de l'interface.

2.1. Recherchez **unity remote.**

3. Sélectionnez la version la plus récente (actuellement 4).

3.1. Téléchargez l'application.

4. Lorsque vous êtes prêt à développer, lancez cette application à partir de votre appareil.

Notez-bien!

Ce programme ne permet pas de tester les performances d'un jeu mobile, mais vous permet d'utiliser votre appareil pour les différents senseurs et l'écran tactile. Ce n'est pas tous les appareils Android qui peuvent être utilisés avec Unity.

***Par exemple**, la tablette Dell Venue 7 n'est pas reconnue par Unity, vérifiez alors avec votre manufacturier avant d'investir dans un achat.*

En retour, le iPhone 6 fonctionne sous Windows et OS X sans problème, même si le projet est développé pour Android. Cependant, cela semble contredire les indications d'Unity qui mentionnent que les appareils iOS ne fonctionnent que sous OS X!

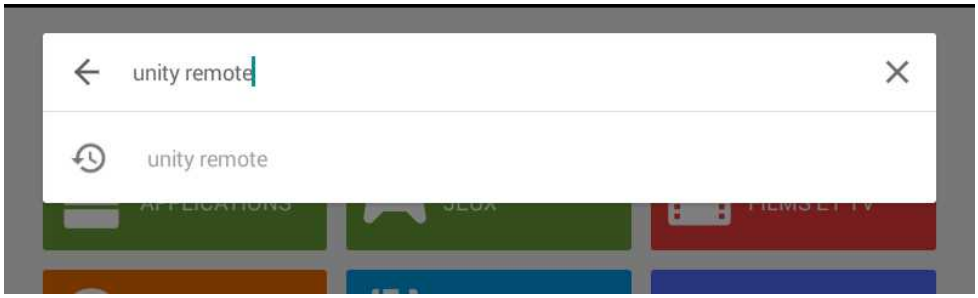
Installation sur Android

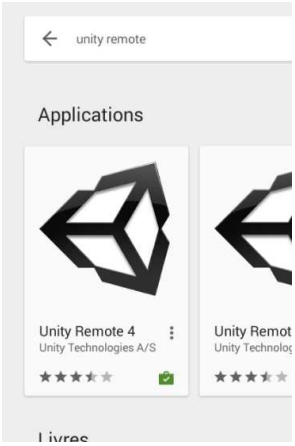
Atelier pratique

1. À partir de votre appareil Android, ouvrez Google Play Store.

2. À partir de Google Play Store, appuyez sur le champ **Search**.

2.1. Recherchez **unity remote.**

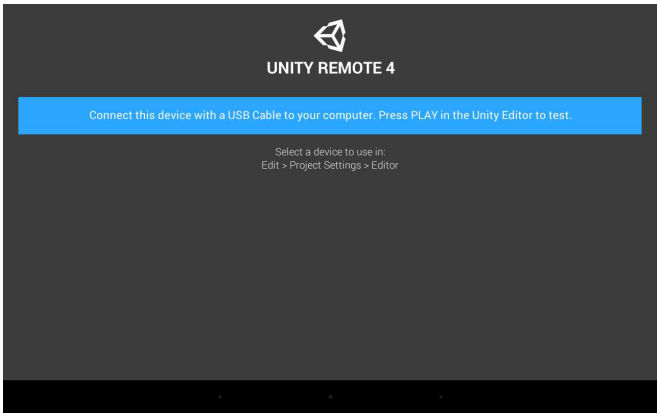


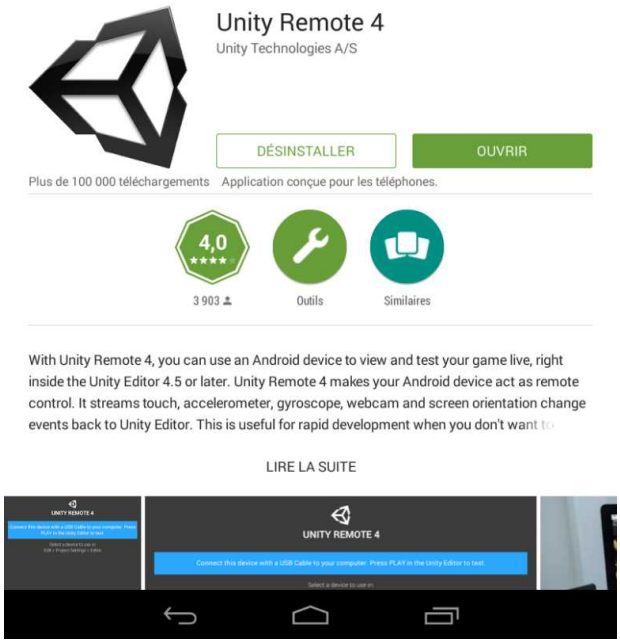


3. Sélectionnez la version la plus récente (actuellement 4).

3.1. Téléchargez l'application.

4. Lorsque vous êtes prêt à développer, lancez cette application à partir de votre appareil.



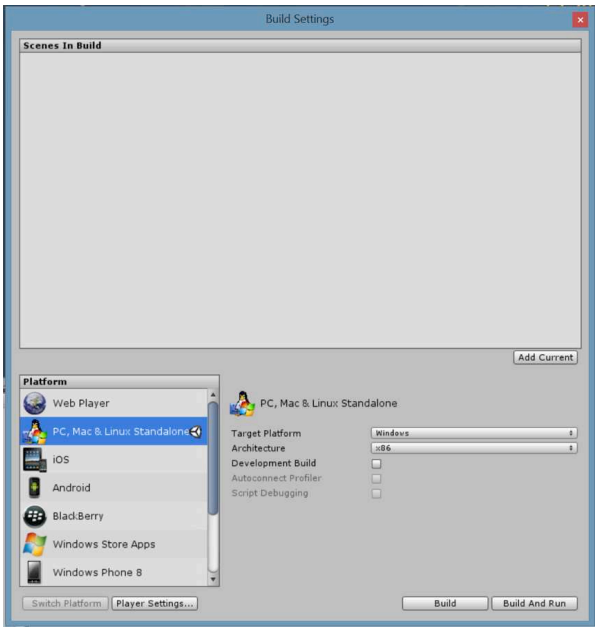


Utilisation de Unity Remote sur PC

Maintenant que votre appareil est prêt, vous devez configurer Unity pour que celui-ci interagisse avec lui.

Atelier pratique

Préparer la plateforme de développement



1. **Ouvrez** ou **créez** un projet dans l'éditeur Unity.

2. À partir de la barre de menus, allez :

File | Build Settings…

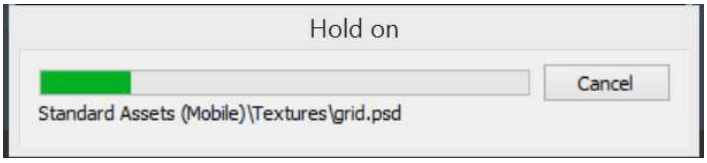
Raccourcis : (⇧**+CTRL+B**) ou (⇧**+⌘+B**) sur Mac.

3. Dans la fenêtre *Build Settings*, vous avez une section nommée ***Platform.***

3.1. Sélectionnez la plateforme pour **iOS**, **Android** ou une autre plateforme mobile.

3.2. Appuyez sur le bouton « Switch Platform ».

Unity va convertir les fichiers pour la nouvelle plateforme.



Une fois la plateforme changée, vous devriez voir l'icône d'Unity à côté de son nom.

Pour l'instant, vous pouvez fermer cette fenêtre.

Importer le *package* mobile

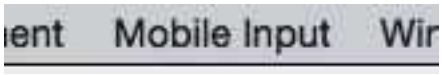
Maintenant que la plateforme est sélectionnée, vous allez importer les outils mobiles dans *Project*.

4. À partir de la barre de menus, allez dans :

Assets | Import Package | CrossPlatformInput

4.1. Importez le contenu du package.

Sauvegardez votre scène (CTRL+S) ou (⌘+S) sur Mac.



Vous devriez avoir un nouveau menu nommé **Mobile Input** dans votre barre de menus.

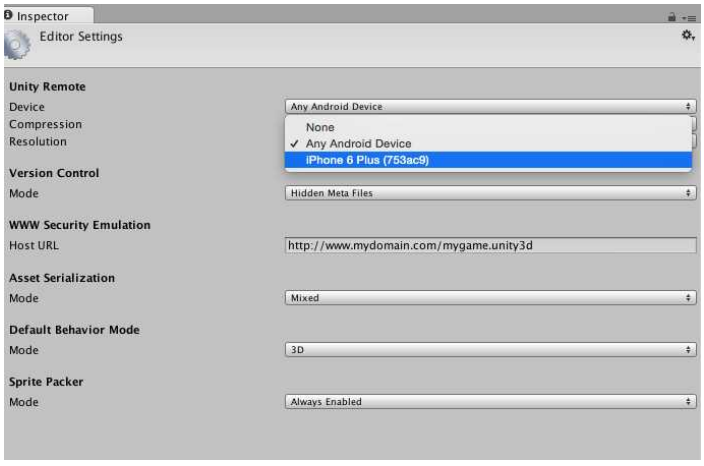
Utilisez les scripts et les exemples qui sont fournis pour créer votre interface et vos contrôles mobiles. Lorsque vous êtes prêts à tester…

Activation de Unity Remote

5. Branchez votre appareil mobile sur votre ordinateur avec son câble USB.

6. Ouvrez l'application **Unity Remote** sur votre appareil mobile.

Retournez à l'éditeur



7. À partir de la barre de menus, allez :

Edit | Project Settings | Editor

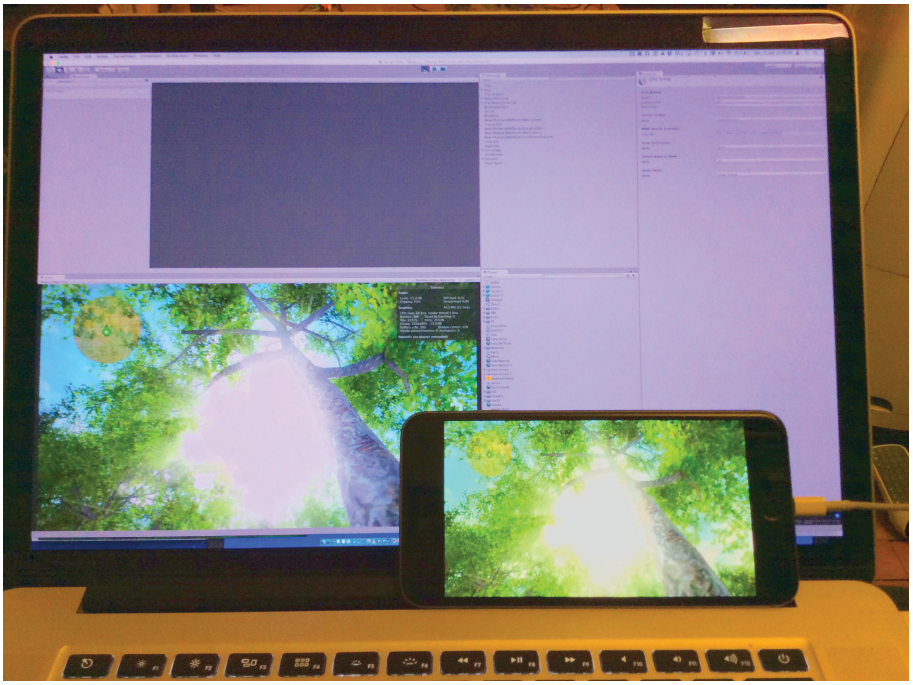
7.1. Dans votre *Inspector*, changez :

Unity Remote
7.1.1.***Device*** : (votre appareil dans la liste)

Notez-bien!

Si votre appareil ne figure pas dans cette liste, c'est qu'il n'est pas reconnu par l'éditeur. Veuillez-vous assurer que vous avez les pilotes de votre appareil pour PC. Sur Mac, les pilotes Android ne sont pas toujours disponibles. Il est donc recommandé d'utiliser des appareils iOS ou des appareils Android qui sont certifiés pour fonctionner sur Mac. Vous pouvez également exposer votre problème sur les différents Forums de discussions sur le Web.

Lorsque vous êtes prêt à tester, vous n'avez qu'à appuyer sur le bouton « Play » et votre appareil devrait afficher le jeu.



↘ *Quand tout se passe bien, voici ce que vous devriez obtenir.*

Vous pouvez désormais utiliser les différentes technologies de votre portable lors du développement.

Lorsque le jeu est prêt à être testé sur l'appareil, il faut quand même savoir comment le déployer sur les différentes plateformes.

Comment compiler pour Android

Atelier pratique

Dans cet exercice, vous allez installer et configurer votre poste de travail afin de créer un jeu pour *Android*.

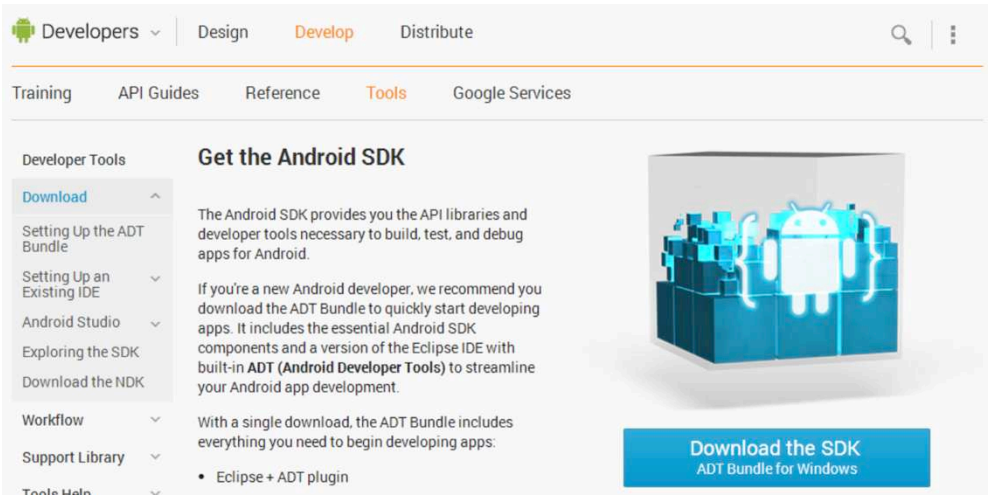
Afin d'installer le *SDK d'Android,* il est essentiel de télécharger et d'installer une version de Java 7 sur votre ordinateur (JRE seulement ne suffit pas).

1. Allez sur le site de téléchargement de *Java* et téléchargez le *JDK* pour votre système:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

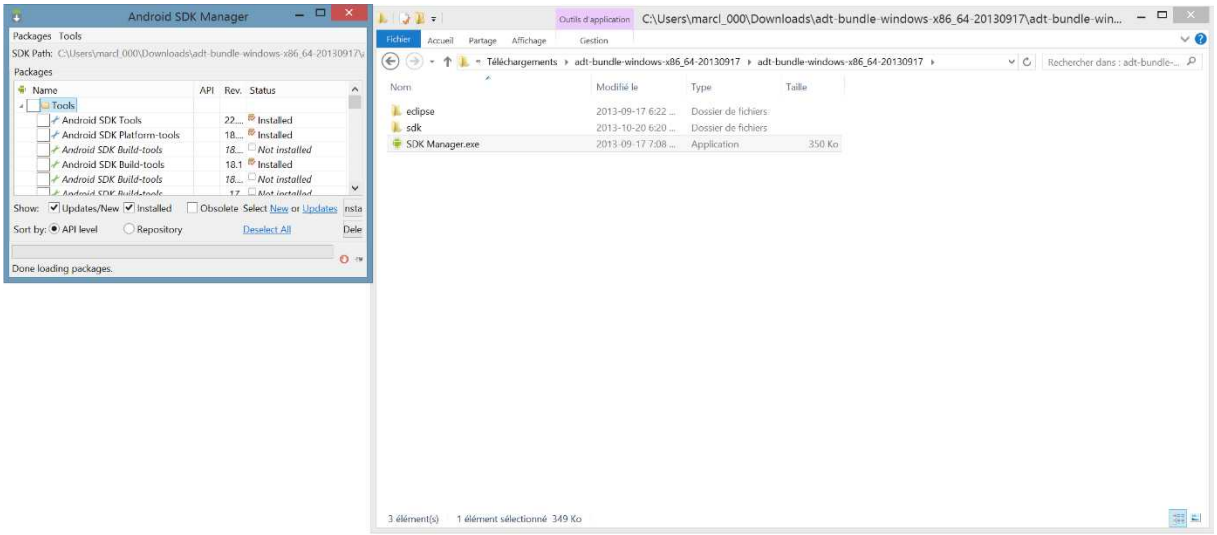
2. Lorsque le *JDK* est installé, veuillez récupérer l'*Android SDK* du site de développeurs de *Google*. (~500Mo)

http://developer.android.com/sdk/index.html



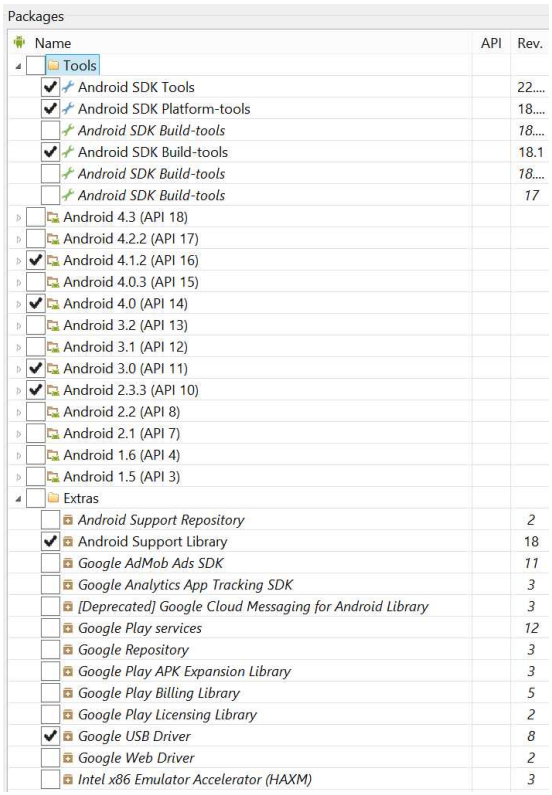
Lorsque les fichiers du *SDK* sont décompressés :

3. Lancez le programme ***SDK Manager*** sur votre ordinateur pour télécharger les *API* et les outils de développement nécessaires pour *Unity*.



La majorité des appareils utilise Android 4.1 comme base pour leur système d'exploitation afin de maximiser le potentiel du jeu.

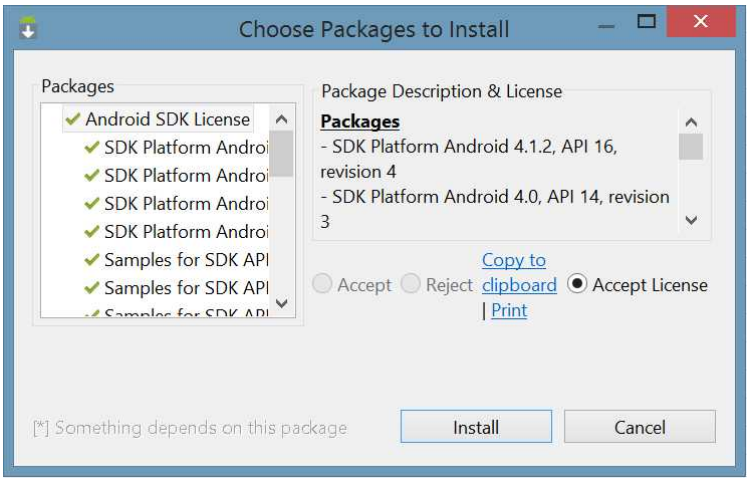
4. Utilisez « l'API 16 », si vous voulez distribuer le jeu sur la majorité des appareils.



↘ *Récupérez les paquets par défaut et ajoutez Android 4.1.2 (API 16)*

5.Appuyez sur le bouton « Install packages » pour débuter le téléchargement.

6.Dans le panneau *Choose Packages to Install,* **cochez** sur « Accept License », puis appuyez sur le bouton « Install ».



Lorsque le SDK est installé sur votre poste de travail :

Exporter sur Android dans Unity®

1. Ouvrez ***Unity*** ainsi que votre projet pour l'exportation.

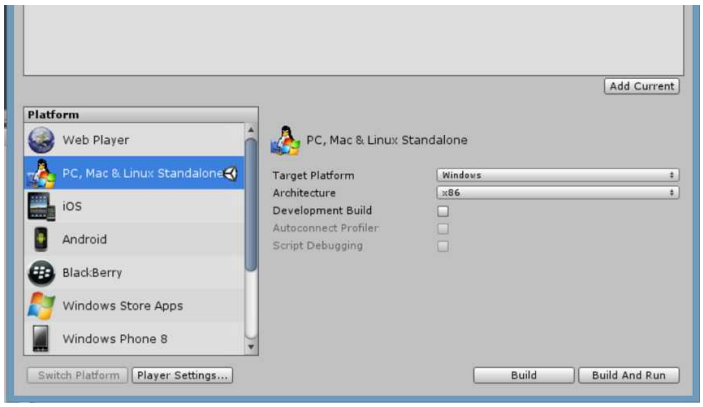
Vous allez changer et configurer la plateforme pour *Android*.

2. Allez dans le menu :

File | Build settings…

Raccourcis : (**CTRL+⇧+B**) ou (**⌘+⇧+B**) sur Mac.

Par défaut, aucune scène n'est présente dans le ***build***.

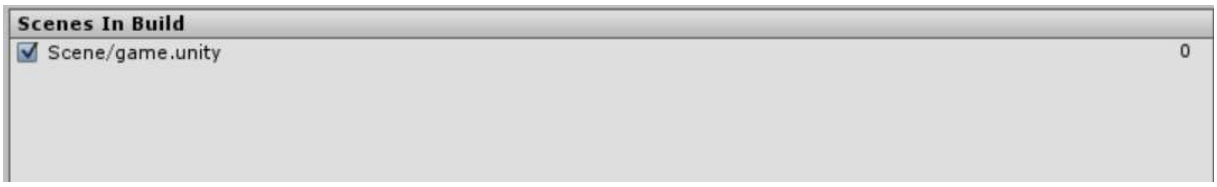


3. Appuyez sur le bouton « Add Current » pour mettre la scène actuelle dans la liste.

Vous devriez voir la scène dans la liste.

Notez-bien!

Vous pouvez mettre plusieurs scènes dans la liste. Elle indique l'ordre dans lequel les scènes vont s'enchaîner.

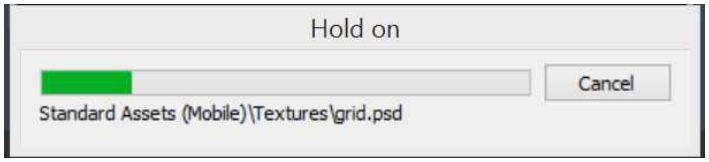


Vous allez maintenant changer la plateforme pour *Android*.

4. Dans la section ***Platform***, sélectionnez **Android.**

4.1. Appuyez sur le bouton « Switch Platform ».

Unity va convertir les fichiers pour la nouvelle plateforme.

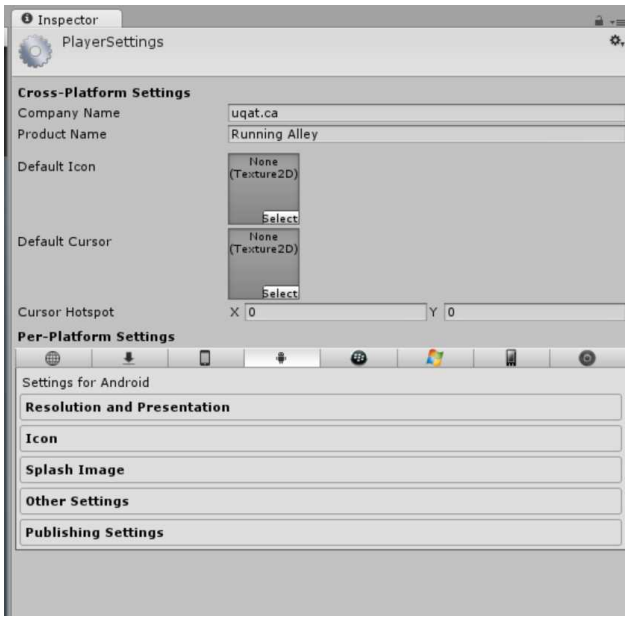


Lorsque la plateforme est changée, vous devriez voir l'icône d'Unity à côté du titre *Android*.

Vous devez maintenant configurer la plateforme afin que le jeu fonctionne correctement sur celle-ci.

4.2. Appuyez sur le bouton « Player Settings… ».

L'Inspector affiche maintenant les propriétés de la plateforme.



4.3. Dans *l'Inspector*, veuillez définir les informations suivantes :

Cross-Platform Settings
4.3.1. ***Company Name*** : Entrez le nom de votre compagnie sous forme d'une url. Ex.: « *amazon.fr* », « *monDomaine.com* »
4.3.2. ***Product Name*** : Entrez le nom de votre produit. Ex. : **Running Alley**, **Mon Super Jeu**…
4.3.3. ***Default Icon*** : Insérez une icône pour votre jeu.

Pour les éléments spécifiques à *Android* :

5. Allez dans la section ***Per-Platform Settings*** (la quatrième icône) à partir de la gauche.

Pour répondre aux exigences d'Android, vous allez utiliser deux sous-sections : ***Resolution and Presentation*** et ***Other Settings***.

Resolution and Presentation
5.0.1. ***Default Orientation*** : Auto Rotation

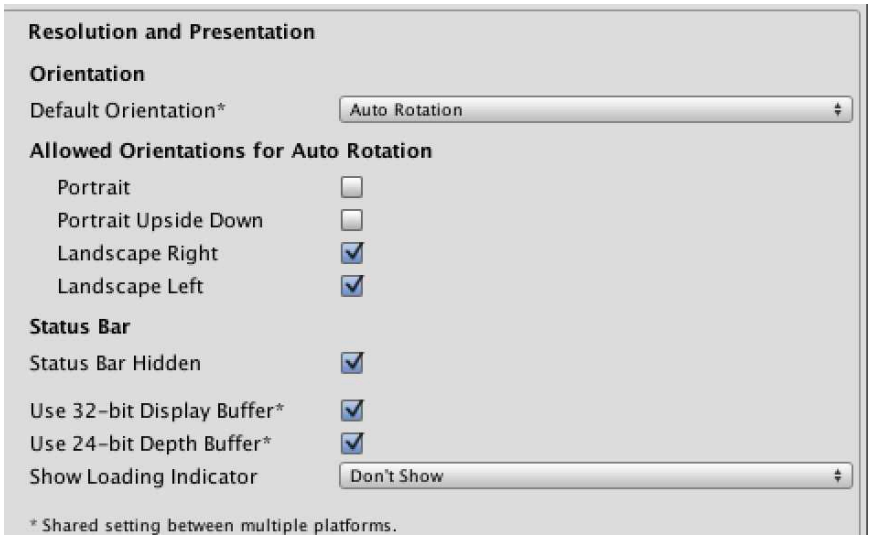
Allowed Orientations for Auto Rotation
Si le jeu se joue uniquement en mode ***portrait***, il est recommandé de cocher :

- **Portrait**
- **Portrait Upside Down**

Si le jeu se joue uniquement en mode ***paysage***, alors il est recommandé de cocher :

- **Landscape Right**
- **Landscape Left**

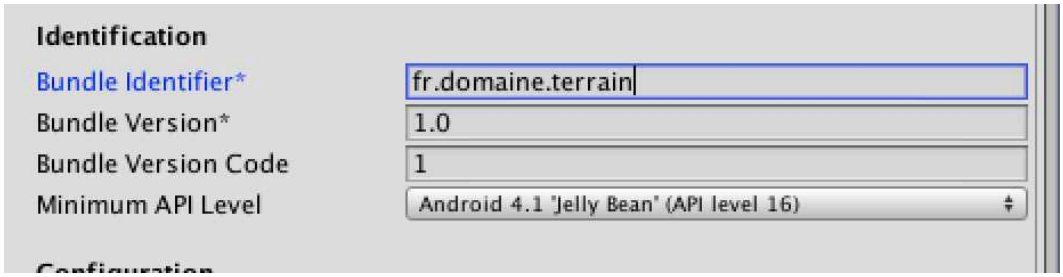
Si le jeu supporte les deux modes, alors cochez tout. Il est important de ne pas présumer que l'usager tient son appareil dans une orientation fixe.



Other Settings

5.0.2. ***Bundle Identifier**** : l'identifiant du paquet devrait être l'*url inversée* de votre compagnie avec le nom du jeu à la fin.
Ex. : « *fr.domaine.nomduJeu* », « *com.company.produit*»…

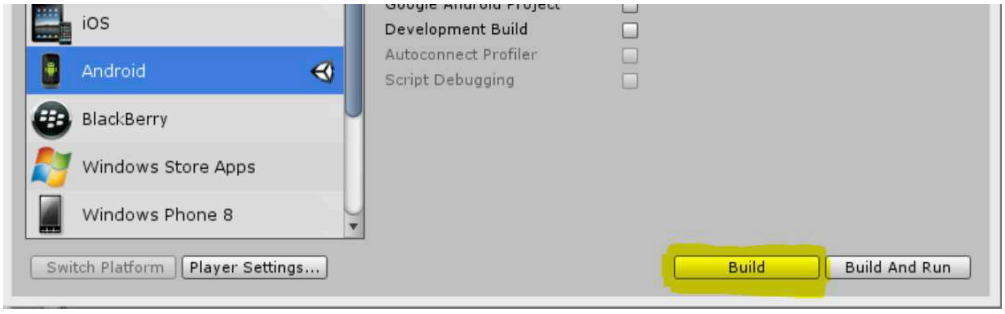
5.0.3. ***Minimum API Level*** **:** Android 4.1 'Jelly Bean' (API level 16)



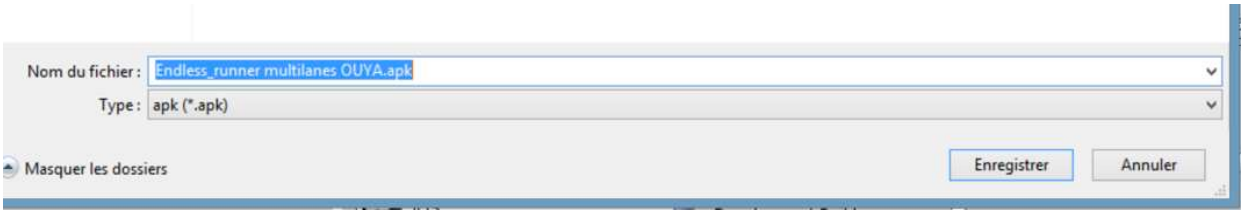
Sauvegardez votre projet.

Vous êtes prêts à compiler le jeu.

6. Dans la fenêtre : *Build settings…* appuyez sur le bouton « Build ».

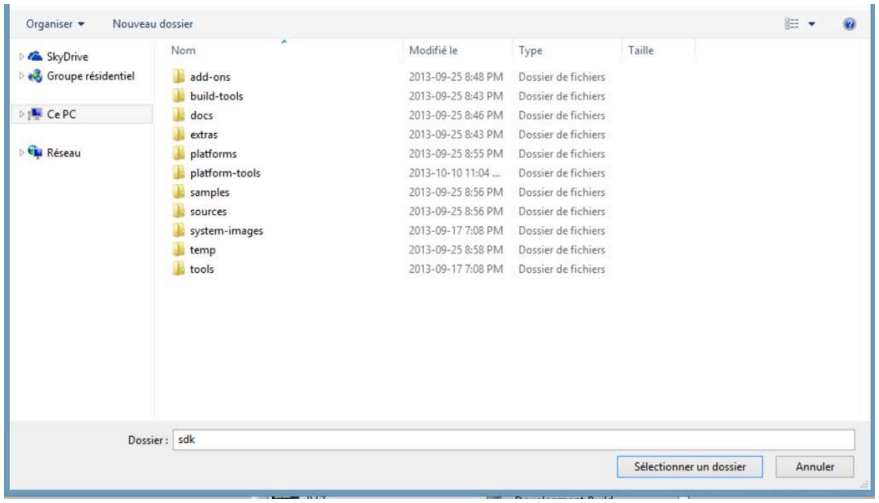


Sauvegardez votre paquet (.APK) sur le disque dur.



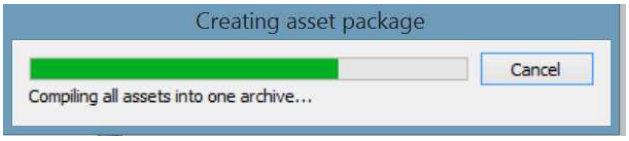
Lors de la première compilation de votre projet, Unity vous demandera de lui spécifier l'emplacement d'Android SDK sur votre ordinateur.

7. Sur Windows, sur *Finder* ou sur Mac, dans *l'explorateur,* naviguez sur le disque dur pour trouver le répertoire du *SDK d'Android* que vous avez déjà installé.



8. Une fois trouvé, appuyez sur le bouton : « Sélectionner un dossier » ou « Ouvrir » sur Mac.

Unity complètera la compilation.



Voici maintenant un fichier (.APK) que l'on peut installer sur Android.

La théorie sur la compilation, pour *Android,* est maintenant complétée. Unity ne vous redemandera pas de lui spécifier où se trouve le *SDK d'Android*, à moins que celui-ci soit supprimé.

Installation du jeu

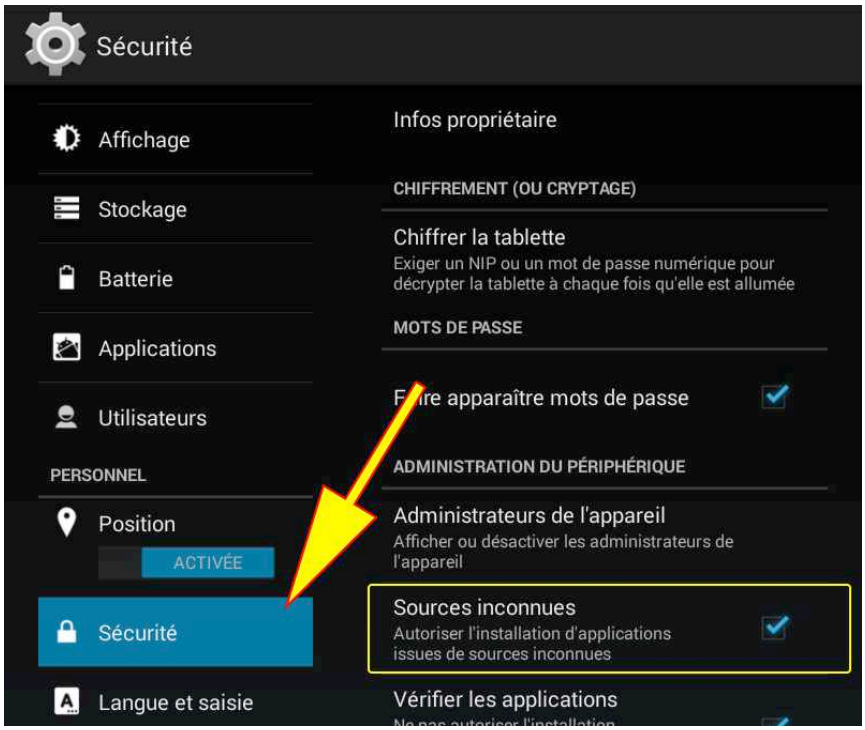
Pour installer le programme sur Android, vous devez d'abord transférer le fichier (.APK) sur votre appareil.

La méthode simple est d'utiliser **Google Drive** (ou un autre espace de disque dans le nuage).

1. À partir de votre navigateur Web préféré, allez vous connecter sur votre compte **Google Drive** (ou votre espace disque dans le nuage de votre choix).

2. Envoyez votre fichier (.APK) sur cet espace disque.

Sur Android, afin de pouvoir installer des apps de sources non-officielles (c'est-à-dire qui ne viennent pas de **Google Play Store**), il faut donner la permission à l'appareil d'installer des programmes de sources inconnues.



Sur l'appareil

3.Allez dans les **paramètres** de votre appareil.

3.1. Puis sélectionnez **Sécurité.**

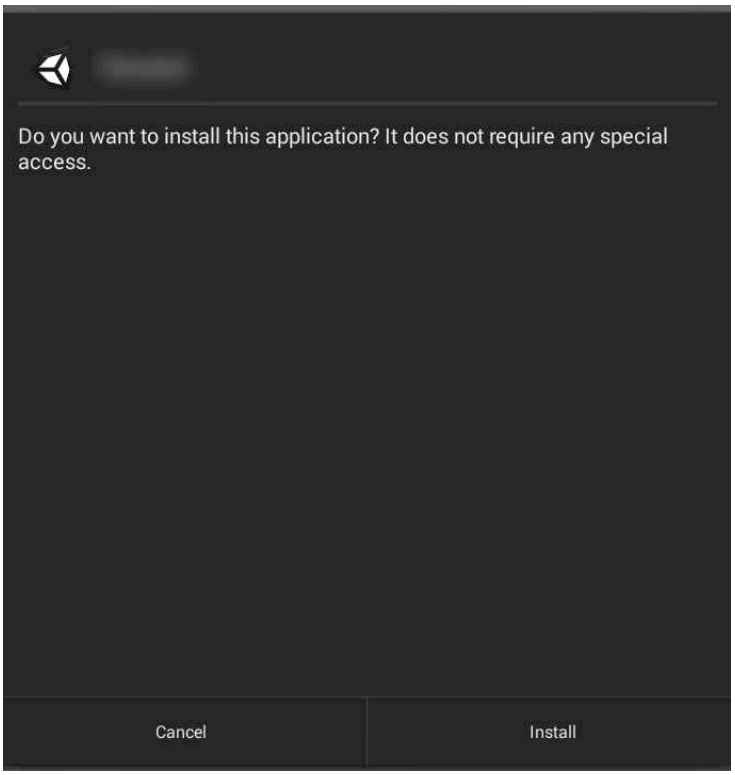
3.1.1. Cochez **Sources inconnues.**

Maintenant que l'on peut installer des applications personnelles, il faut ouvrir le fichier (.APK) à partir de Google Drive.

4.Ouvrez Google Drive app, (ou l'apps de l'espace en ligne que vous utilisez).

4.1. Cliquez sur le fichier .APK pour le télécharger.

5. L'installation devrait débuter lorsque vous appuyez sur le bouton « Install ». Vous n'avez qu'à suivre les instructions à l'écran.



Ce n'est pas plus compliqué que cela. Maintenant enchaînez avec la plateforme de la pomme californienne !

Comment compiler pour Apple iOS

!!!!! Requiert un ordinateur Apple avec OS X 10.9 ou plus récent !!!!!

Atelier pratique

Dans cet exercice, Vous allez installer et configurer un poste de travail pour créer un jeu pour *Apple iOS* (iPhone, iPad, iPod Touch).

Vous devez absolument posséder un ordinateur Mac afin de compléter la compilation d'un jeu avec Unity 5.

Les *hackintoshes* (clones Mac) ne seront pas autorisés pour publier un titre. Par contre, vous pouvez faire le développement complet sur PC et même exporter un projet *Xcode* avec Unity® sous *Windows*.

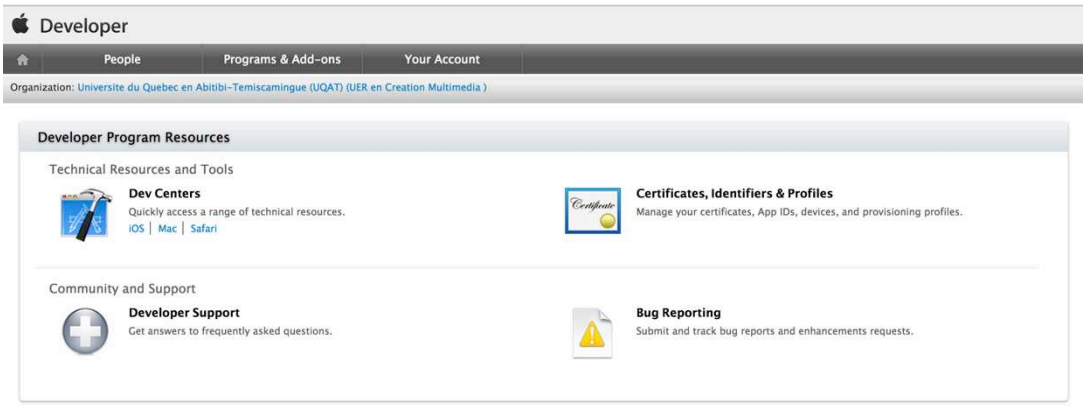
Pour débuter vous devez vous inscrire comme développeur *iOS* sur le site d'*Apple*.

https://developer.apple.com/programs/ios/

(Le site de développement est en anglais seulement).

Développer pour Apple coûte moins de 100$ US par année. Une fois inscrit, vous pouvez débuter le développement.

Une fois inscrit, vous pouvez débuter vos préparatifs sur le site ***Developer Program Ressources.***



Vous allez télécharger et installer ***Xcode SDK*** sur le *Mac Apps Store.*

1. Ouvrez le ***Mac Apps Store*** sur votre Mac.

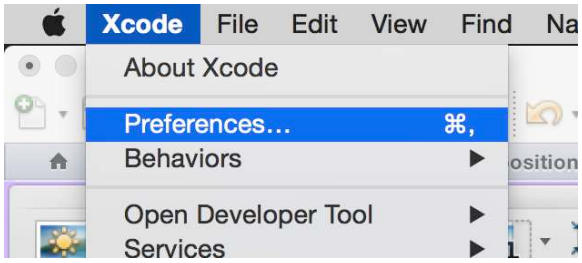
1.1. De là, recherchez **Xcode** ou allez dans les apps **essentials** d'*Apple.*

1.2. Téléchargez le *SDK* (~2.49 Go).

Les fichiers s'installent automatiquement, il n'y a aucune maintenance à faire car *Apple* s'en charge.

2. Ouvrez ***Xcode***, il est probable qu'aucune fenêtre ne s'ouvre par défaut quand il n'y a aucun projet ouvert.

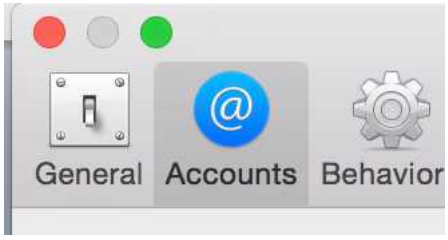
Assurez-vous que le mot ***Xcode*** apparaisse dans la barre de menus. (***Xcode*** est en anglais uniquement).



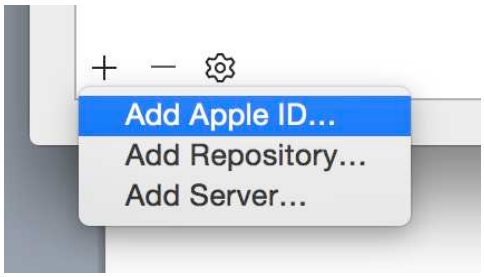
3.Dans la barre de menus, cliquez sur :

Xcode | Preferences...

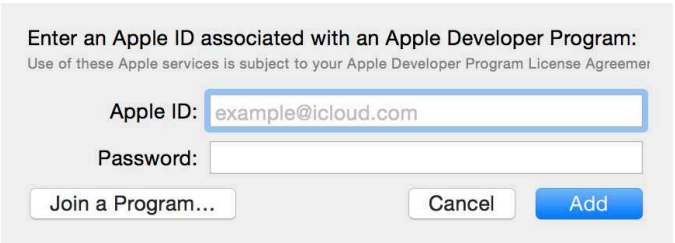
4. Dans la fenêtre *Preferences*, cliquez sur l'onglet « Accounts **».**



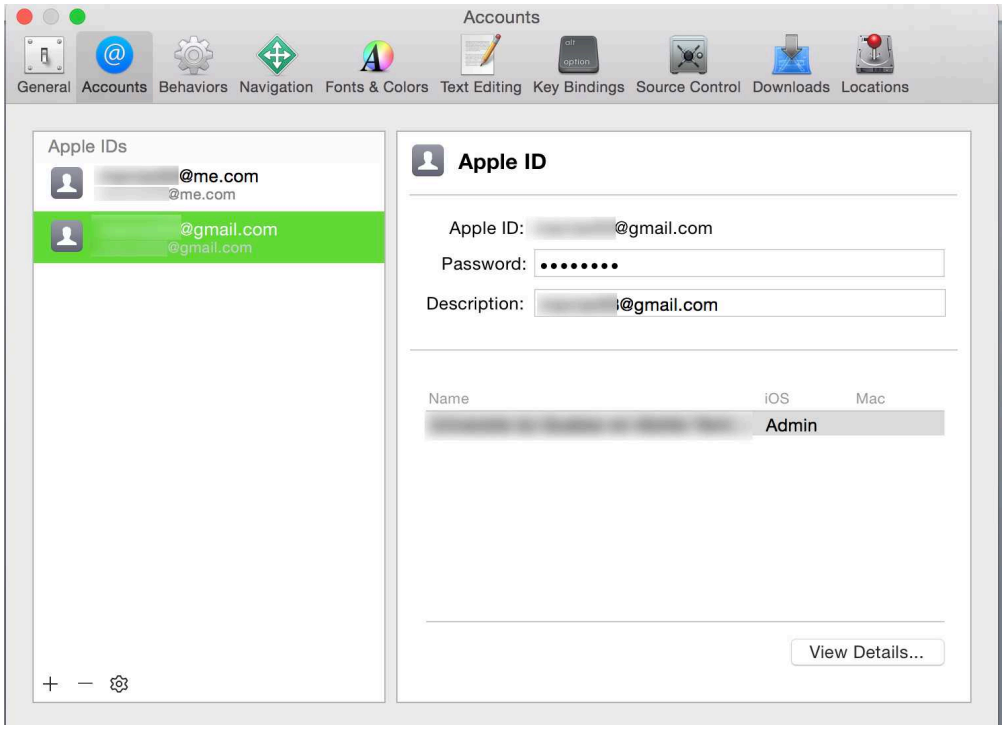
5. Ajoutez votre identifiant *Apple* en appuyant sur le **(+)** dans le coin en bas à gauche.



6. Si vous n'êtes pas déjà inscrit dans le programme de développeurs, vous pouvez le faire en appuyant sur le bouton « Joins a Program... ». Sinon, appuyez tout simplement sur « Add ».

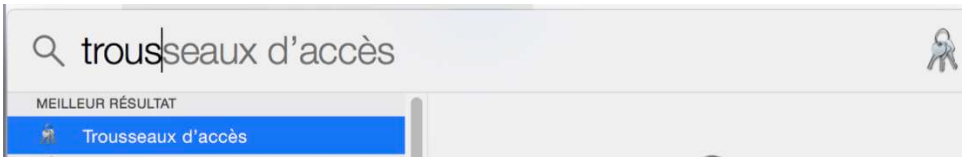


Une fois que vous êtes connecté, vous devriez voir votre statut de *développeur iOS* dans votre fiche ***Apple ID.***



La prochaine étape est de configurer votre ordinateur Mac avec Apple afin de pouvoir compiler des apps localement pour vos appareils, sans utiliser les serveurs d'authentification d'Apple.

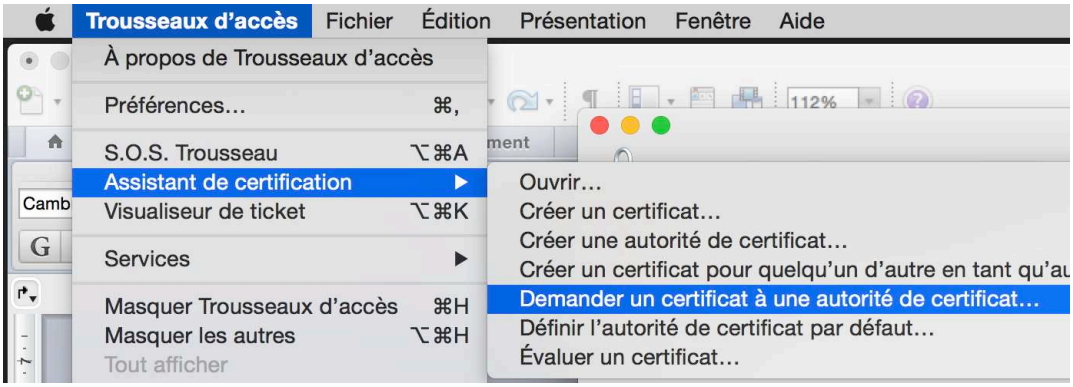
7. Sur votre Mac, ouvrez l'application ***Trousseaux d'accès***.



Ce programme peut sembler complexe, mais la prochaine étape est triviale et doit être configurée seulement une fois par Mac.

8. Dans la barre de Menus ***trousseaux d'accès,*** allez dans :

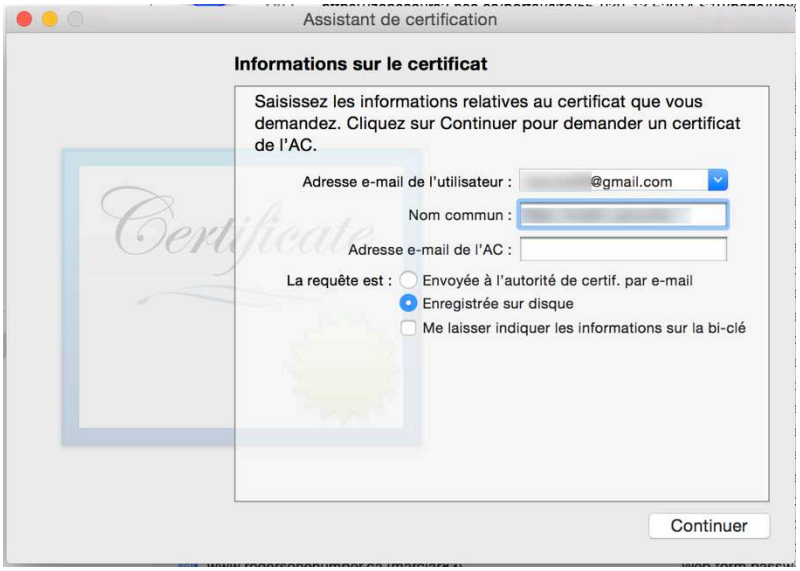
Trousseaux d'accès | Assistant de certification | Demander un certificat à une autorité de certificat...



L'assistant de certification va s'ouvrir.

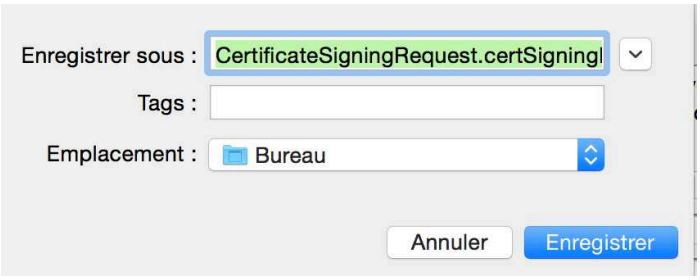
8.1. Entrez les informations suivantes :

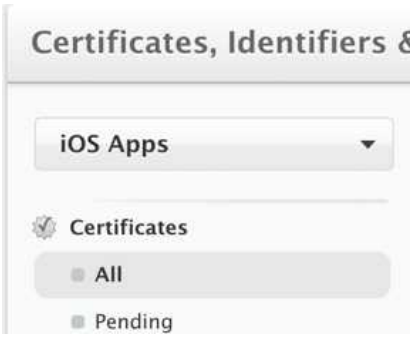
8.1.1. ***Adresse email de l'utilisateur*** **:** [L'adresse avec laquelle vous êtes enregistré].
8.1.2. ***Nom commun*** **:** [Nom de référence (celui-ci n'est pas important)].
8.1.3. ***Adresse email de l'agent de certification*** **:** Laissez ce champ vide.
8.1.4. ***La requête est*** **:** Enregistrée sur disque



8.2. Appuyez sur le bouton « Continuer ».

8.3. Enregistrez la requête de certification (***.certSigningRequest***) sur votre ordinateur.



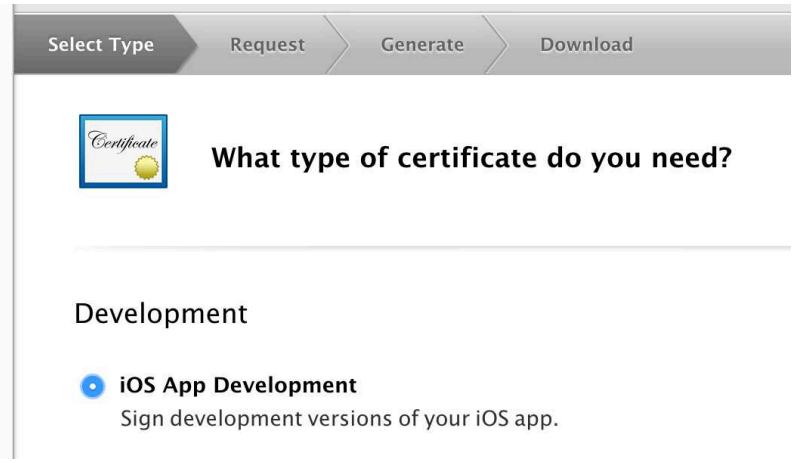


9. Retournez sur le *site de développeurs d'Apple* dans la section :

Certificates | All

10. Appuyez sur le **(+)** qui se trouve dans le coin en haut, à droite de la fenêtre (***iOS Certificates***).

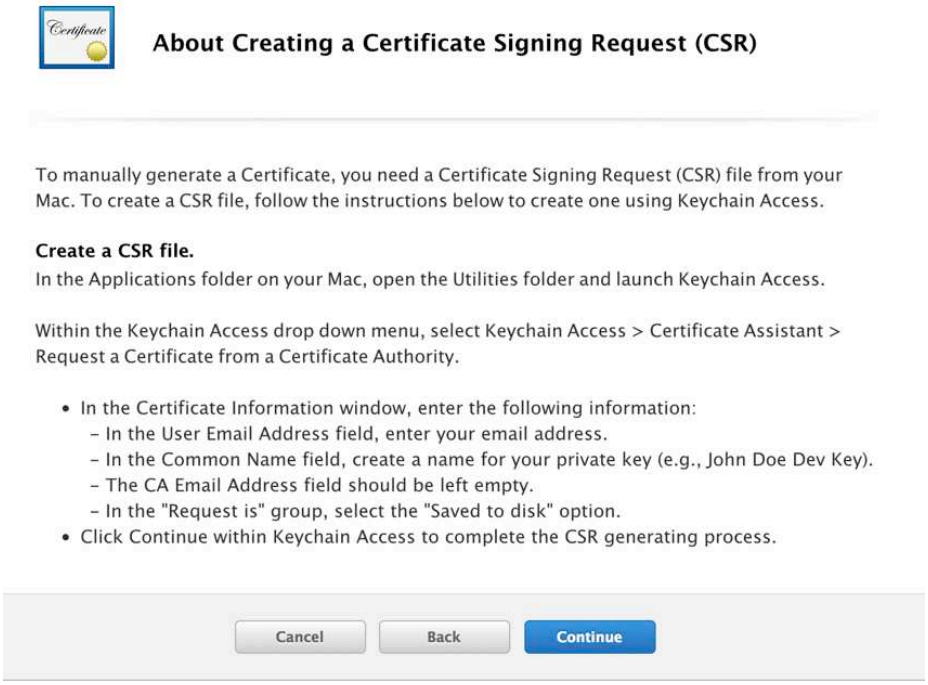
10.1. De-là, vous devez spécifier que vous voulez un certificat de *développement d'app iOS* **(iOS App Development).**



10.2. Appuyez sur le bouton « Continue ».

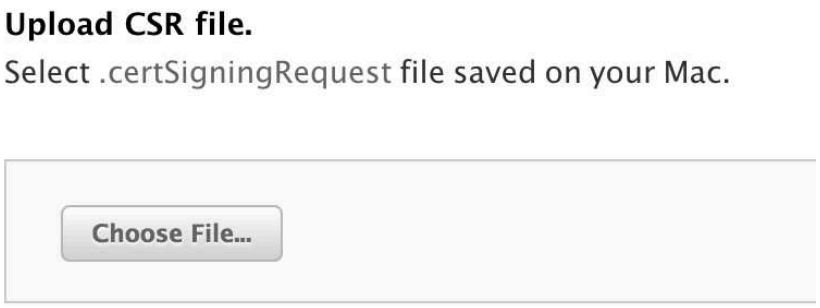
Vous serez invité par Apple à générer la requête avec les instructions pour accomplir cette tâche. Heureusement pour vous, cette étape est déjà complétée.

10.3. Appuyez sur le bouton « Continue » une seconde fois.



La nouvelle fenêtre permet de transférer votre requête de certification à *Apple*.

11. Appuyez sur « Choose File… » et sélectionnez le fichier que vous avez créé précédemment.



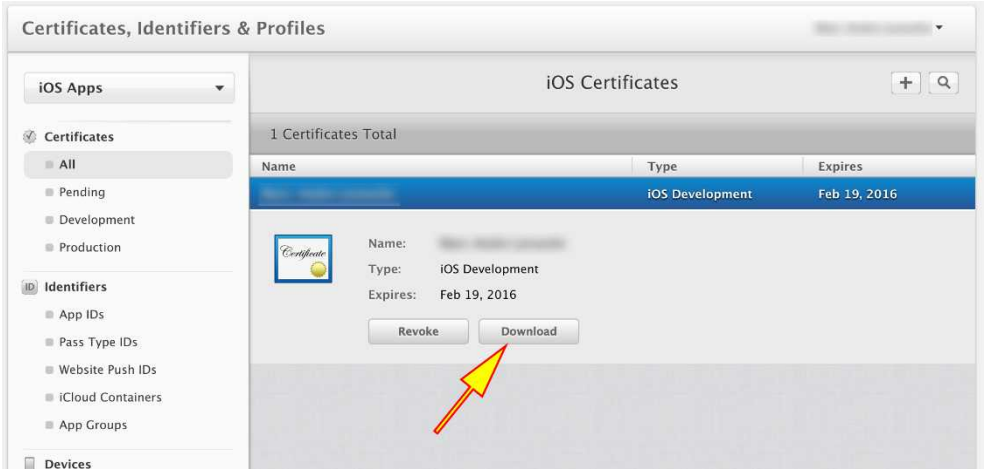
12. Appuyez sur le bouton « Continue » afin qu'*Apple* vous émette votre certificat.

13. Après avoir complété cette étape, vous retournerez à la section ***iOS Certificates***.

Un nouveau certificat devrait figurer dans la liste.

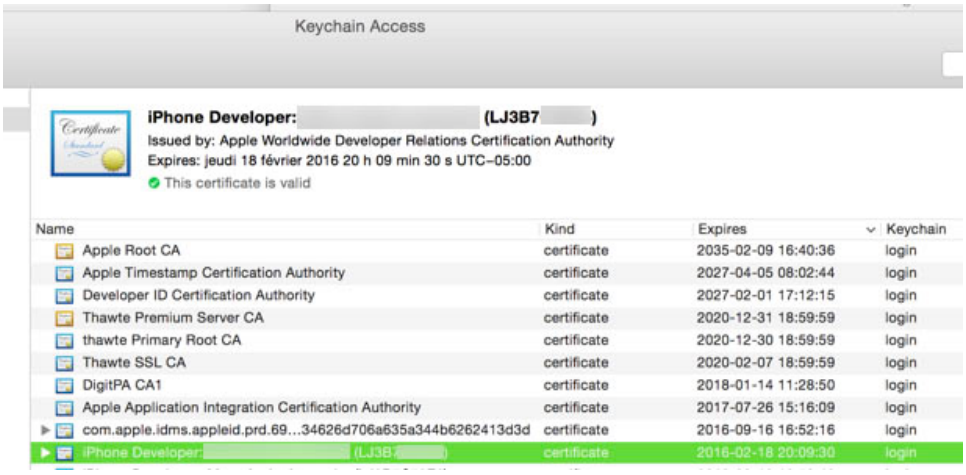
14. Appuyez sur **ce certificat** afin de voir ses détails.

14.1. Appuyez fortement sur le bouton « Download » afin de récupérer votre certificat autorisé.



15. Vous avez un certificat valide sur Mac. Cliquez sur celui-ci dans ***Finder*** afin de l'ajouter à ***trousseaux d'accès***.

Lorsque votre certificat est en place, vous êtes prêt à créer votre premier projet.



16. Retournez sur *le site de développeurs d'Apple.*

16.1. Allez dans la section :

Identifiers | App IDs

16.2. Appuyez sur le **(+)** qui se trouve dans le coin en haut, à droite de la fenêtre : (***iOS App IDs***).

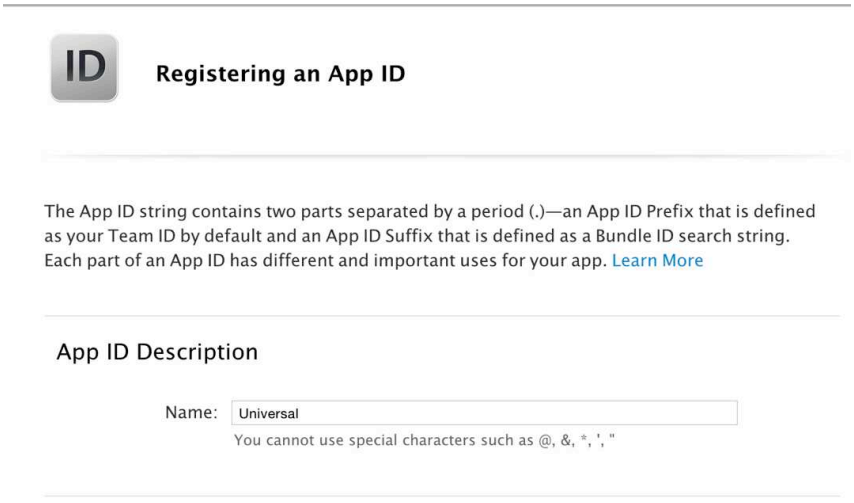
Vous êtes maintenant prêt à créer un ***provisionning*** pour une ou plusieurs apps.

Conseil!

Pour faciliter le prototypage, il est suggéré de créer un identifiant de type « **wildcard** » qui est universel.

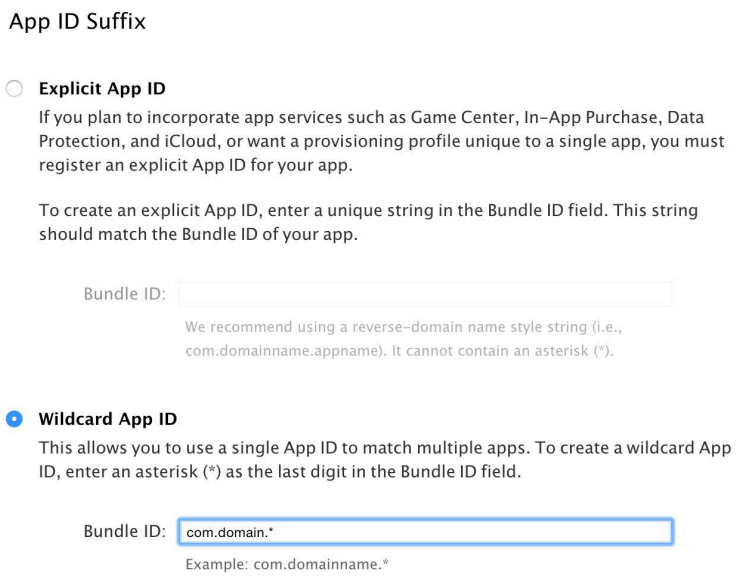
16.3. Dans la section ***App ID Description :***

16.3.1. ***Name*** : Inscrivez **Universel** dans ce champ.



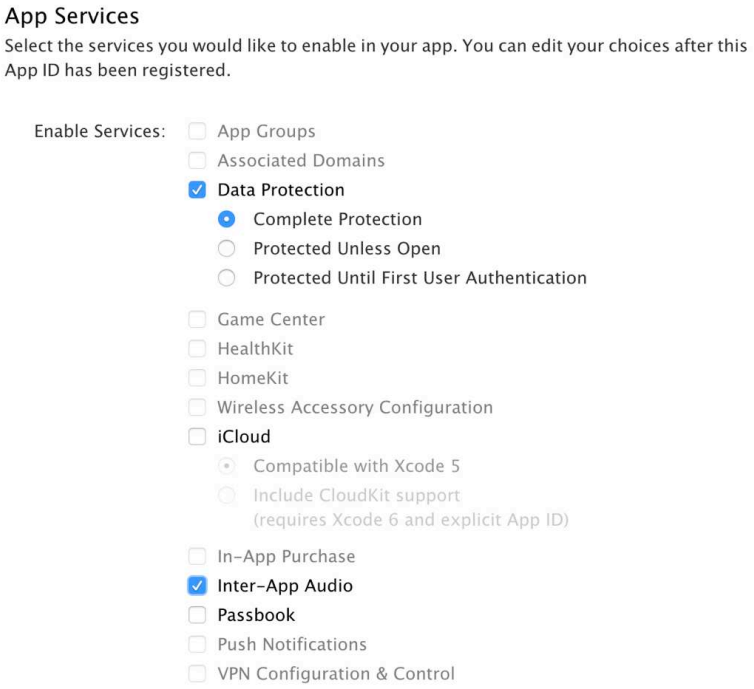
16.4. Dans la section ***App ID Suffix*** :

16.4.1. Cochez : ***Wildcard App ID.***
16.4.2. ***Bundle ID*** : entrez une *URL Inversée* (préférablement l'*URL* de votre site de développeur) en débutant par la fin et en remplaçant les (www) par (*) **:**
ex. : *fr.gamedevcie.*, com.develop.*, ca.domain.**



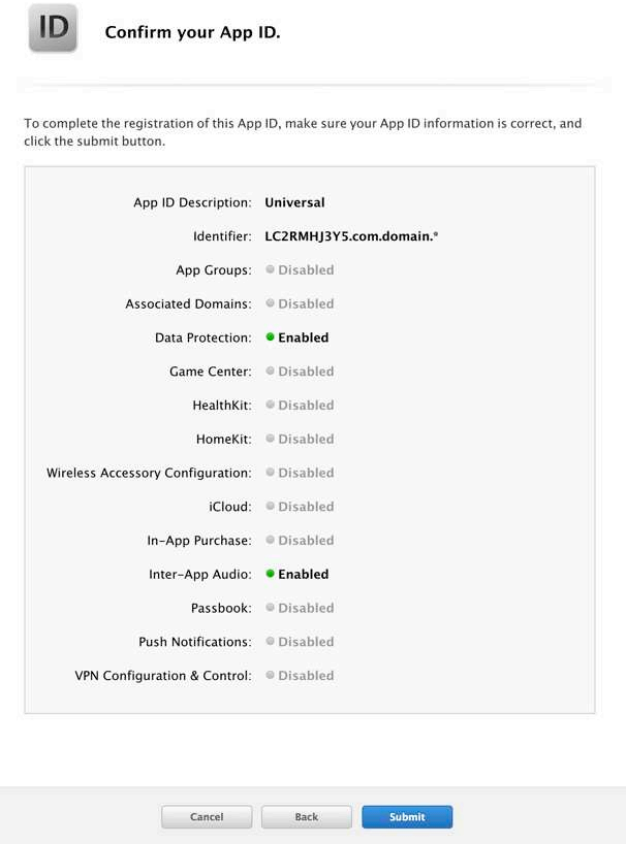
Cet identifiant est très important pour tous vos programmes futurs, notez-le bien afin de le retrouver facilement.

17. Dans la section ***App Services,*** choisissez les services que vous aimeriez utiliser dans vos projets personnels. (Quant à moi, j'utilise une signature ***wildcard***).



18. Une fois complétée, appuyez sur « submit ».

Un ***provisionning*** sera créé.



Notez-bien!

*Les **provisionning** sont des clés temporaires qui certifient les apps avec les appareils enregistrés et le Mac utilisé.*

Vous devez enregistrer vos appareils *iOS* (iPhone *et* iPad) sur le site *d'Apple* afin de fermer la boucle.

Avant de continuer, vous devez récupérer les identifiants des appareils.

19. Branchez l'appareil sur Mac avec un câble USB.

Fermez les logiciels *Photo* et *iTunes* si ceux-ci se sont ouverts d'une façon automatique.

20. Allez dans ***Xcode.***

21. Dans la barre de menus, allez dans :

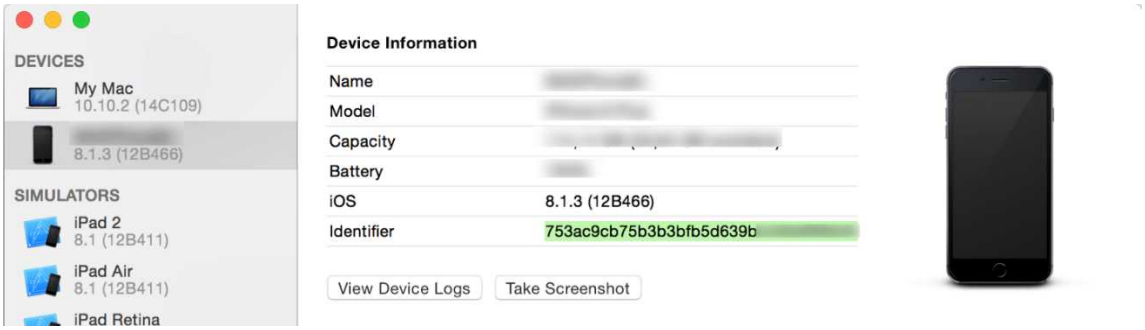
Window | Devices

Dans la liste ***Devices***, vous devriez voir votre appareil.

22. Sélectionnez-le dans la liste.

Dans la section ***Device Information,*** vous devriez avoir une longue chaîne de caractères qui suit le champ ***identifier***.

23. Copiez la valeur d'***identifier***. (C'est le numéro nécessaire pour effectuer la prochaine étape).



Retournez sur le site de *développeurs d'Apple*.

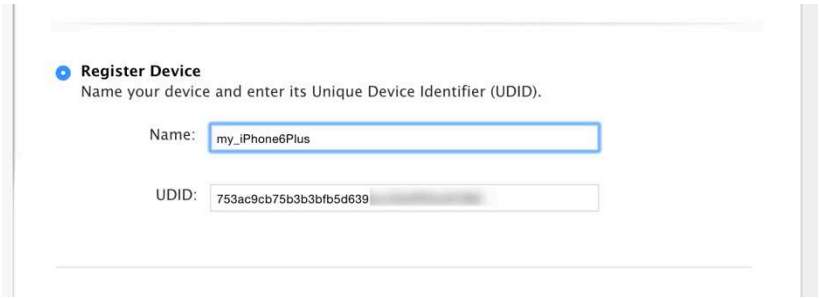
24. Allez dans la rubrique :

Devices | All

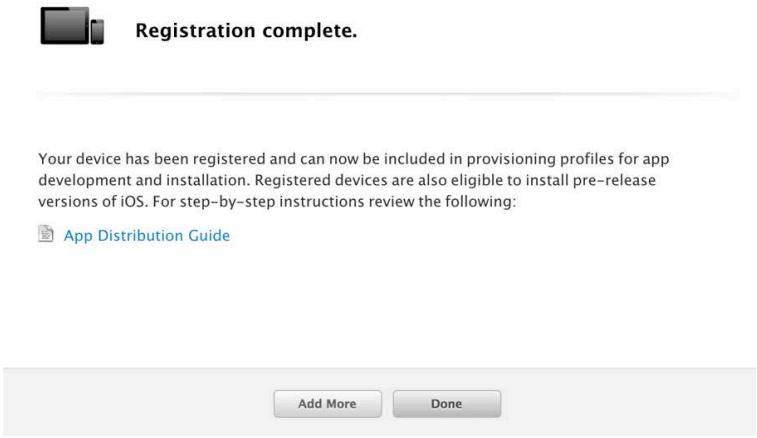
24.1. Appuyez sur le **(+)** qui se trouve dans le coin en haut, à droite de la fenêtre : (***iOS Devices***).

24.2. Dans la section ***Register Device,*** inscrivez :

24.2.1. ***Name*** : Donnez un nom unique à votre appareil.
24.2.2. ***UDID*** : Collez l'identifiant unique que nous avons copié plus tôt.



24.3. Appuyez sur le bouton « Done » pour compléter l'enregistrement.



Exporter sur iOS dans Unity®

Vous allez changer et configurer la plateforme pour iOS.

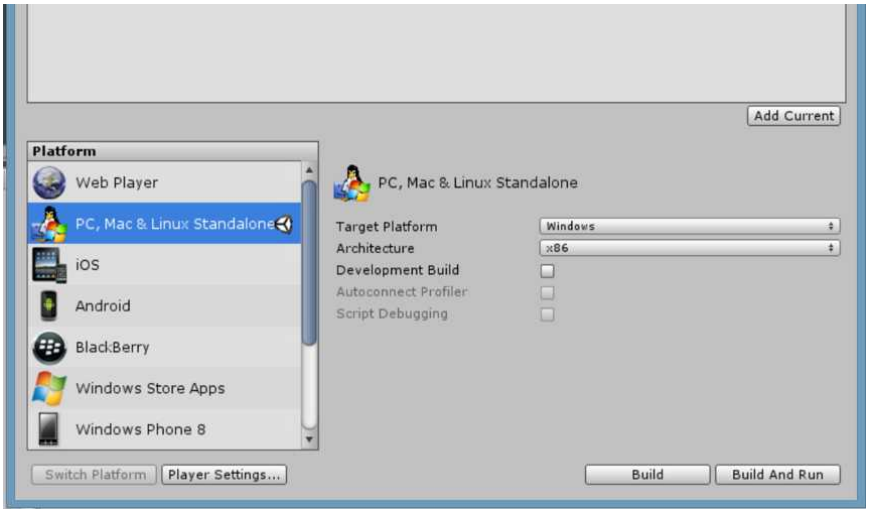
1. Ouvrez **Unity** ainsi que votre projet pour l'exportation.

Vous allez changer et configurer la plateforme pour *iOS*.

2. Allez dans le menu :

File | Build settings...

Raccourcis : (**CTRL+⇧+B**) ou (**⌘+⇧+B**) sur Mac.



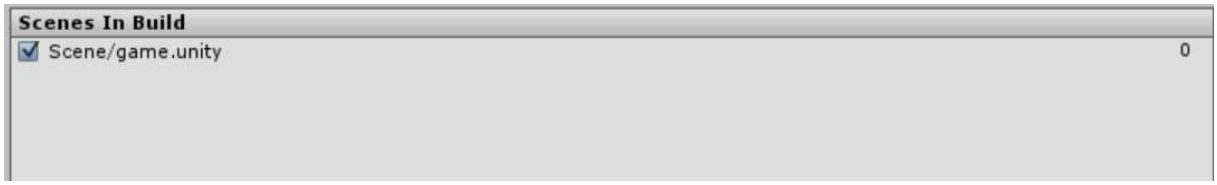
Par défaut, aucune scène n'est présente dans le ***build***.

3. Appuyez sur le bouton « Add Current » pour mettre la scène actuelle dans la liste.

Notez-bien!

Vous pouvez mettre plusieurs scènes dans la liste. Elle indique l'ordre dans lequel les scènes vont s'enchaîner.

Vous devriez voir la scène dans la liste.

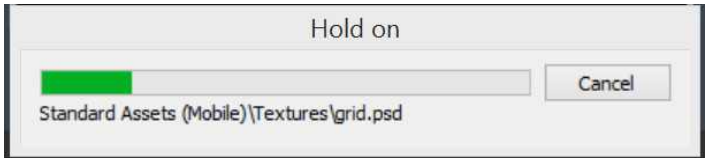


Changez la plateforme pour *iOS*.

4. Dans la section ***Platform***, sélectionnez **iOS.**

4.1. Appuyez sur le bouton « Switch Platform ».

Unity va convertir les fichiers pour la nouvelle plateforme.

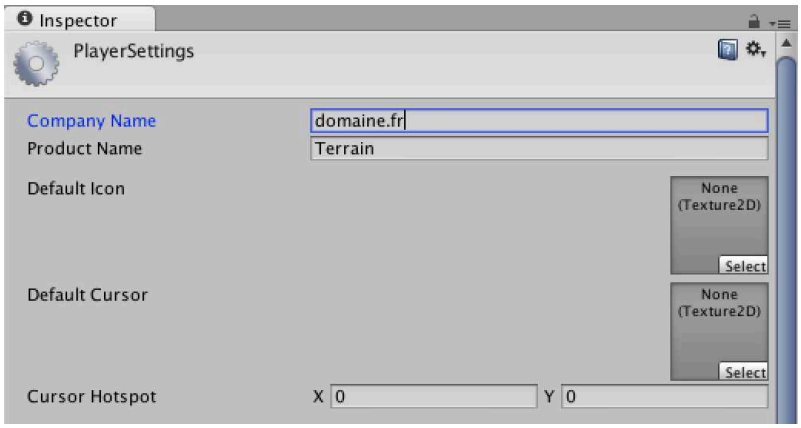


Lorsque la plateforme sera changée, vous devriez voir l'icône d'Unity à côté du titre *iOS*.

Vous devez maintenant configurer la plateforme afin que le jeu fonctionne correctement sur celle-ci.

4.2. Appuyez sur le bouton « Player Settings… ».

L'Inspector affiche maintenant les propriétés de la plateforme.



4.3. Dans *l'Inspector*, veuillez définir les informations suivantes :

Cross-Platform Settings
4.3.1. ***Company Name*** : Entrez le nom de votre compagnie sous forme d'une url. Ex.: « *amazon.fr* », « *monDomaine.com* »
4.3.2. ***Product Name*** : Entrez le nom de votre produit. Ex. : **Running Alley**, **Mon Super Jeu**…
4.3.3. ***Default Icon*** : Insérez une icône pour le jeu.

Maintenant pour obtenir les éléments spécifiques à *iOS* :

5. Allez dans la section ***Per-Platform Settings*** (la troisième icône) à partir de la gauche.

Pour répondre aux exigences, d'iOS, vous allez maintenant utiliser deux sous-sections : ***Resolution and Presentation*** et ***Other Settings***.

Resolution and Presentation
5.0.1. ***Default Orientation*** : Auto Rotation

Allowed Orientations for Auto Rotation

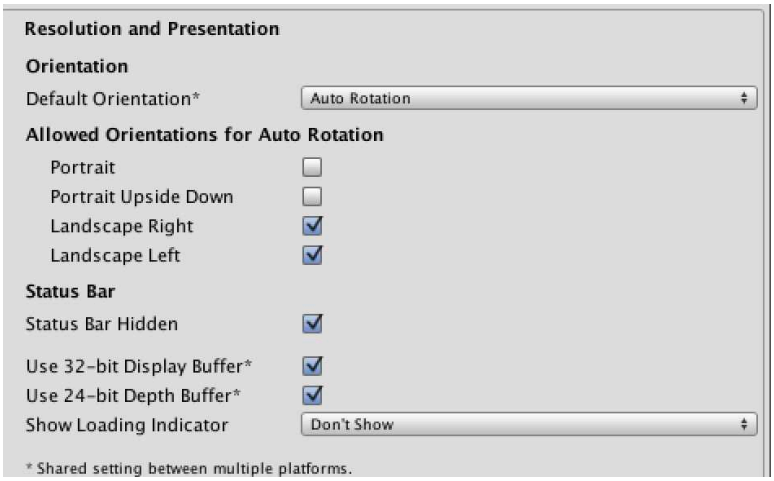
Si le jeu se joue uniquement en mode ***portrait***, il est donc recommandé de cocher :

- **Portrait**
- **Portrait Upside Down**

Si le jeu se joue uniquement en mode ***paysage***, il est donc recommandé de cocher :

- **Landscape Right**
- **Landscape Left**

Si le jeu supporte les deux modes, alors cochez tout. Il est important de ne pas présumer que l'usager tient son appareil dans une orientation fixe.



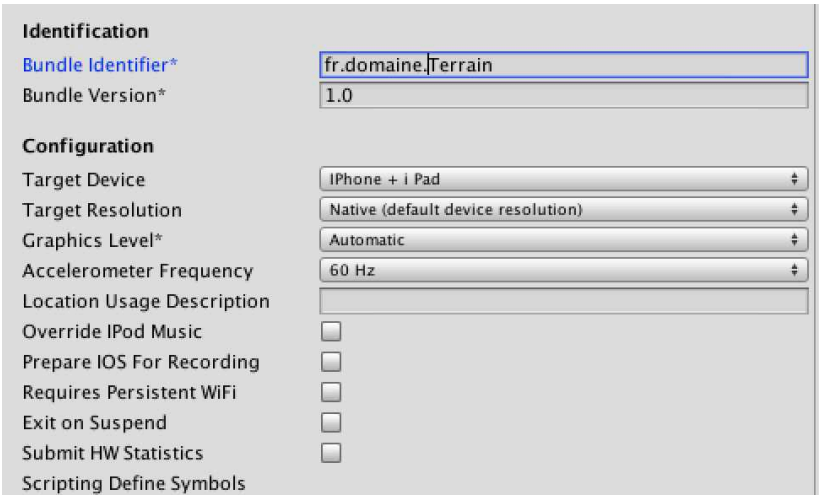
Other Settings

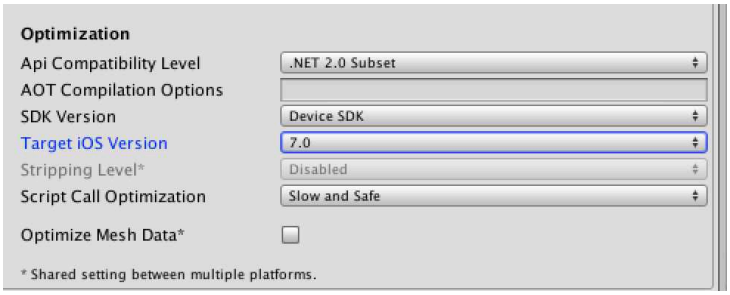
5.0.2. **Bundle Identifier*** : l'identifiant du paquet devrait être l'*url inversée* de votre compagnie avec le nom du jeu à la fin.
Ex. : « *fr.domaine.nomduJeu* », « *com.company.produit*»…

Laissez les autres paramètres tels quels, à moins que vous décidiez :

• De ne pas supporter un type d'appareil (***Target Device***)
• D'afficher le jeu à une résolution autre que la résolution native de l'appareil. (***Target Resolution***)

Les valeurs par défaut sont les valeurs les plus compatibles avec les différents appareils.





Optimization

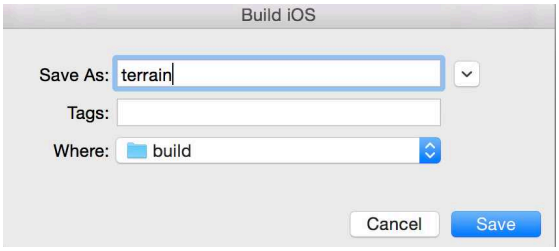
5.0.3. ***Target iOS Version :*** Spécifie que l'on veut supporter la version **7.0** car 97% des utilisateurs actifs utilisent iOS 7 ou un plus récent qui date du 2 février 2015. Veuillez consulter les données les plus à jour pour connaître quelle version d'iOS choisir.

Sauvegardez votre projet.

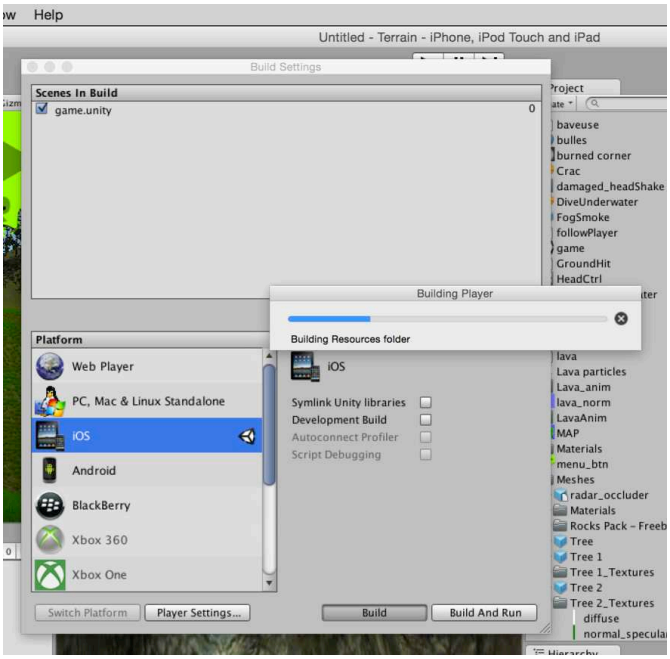
Vous êtes prêt à compiler le jeu.

6. Dans la fenêtre : *Build settings...* appuyez sur le bouton : « build ».

Sauvegardez votre projet (**Xcode**) sur le disque dur.



Unity® va ensuite convertir les fichiers sources en fichiers sources ***Xcode***.



Installation du jeu

1. Ouvrez le répertoire que vous venez d'exporter.

1.1. À l'intérieur de ce répertoire, vous y retrouverez un fichier (**.xcodeproj**), c'est votre jeu.

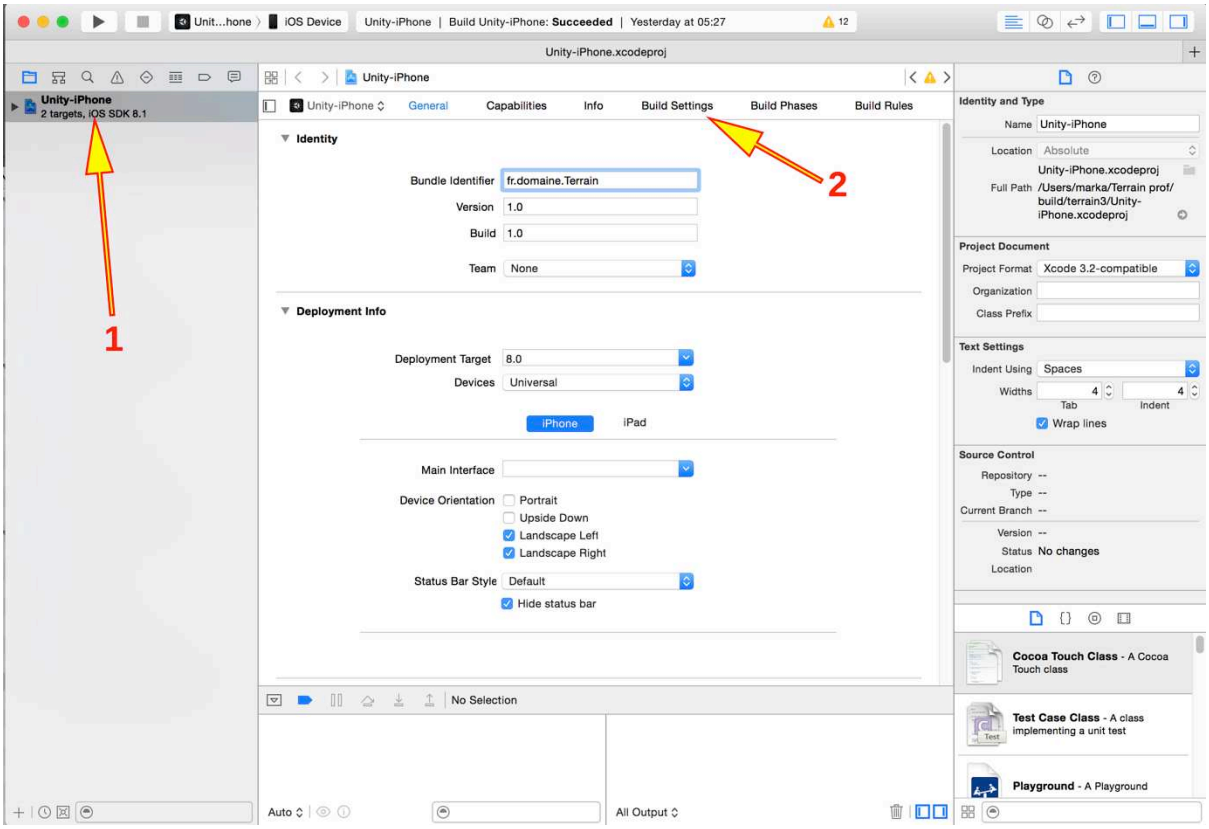
1.2. Cliquez dessus pour l'ouvrir dans **Xcode**.

Pour la première fois, ***Xcode*** s'ouvre avec une fenêtre de projet.

2. Dans la colonne de gauche, sélectionnez votre projet **Unity-iPhone.**

Vous devriez maintenant voir les informations générales de votre jeu : Bundle Identifier, Version, Build, etc.

3. Juste au-dessus de ces informations, il y a une barre d'onglets, cliquez sur « Build Settings ».

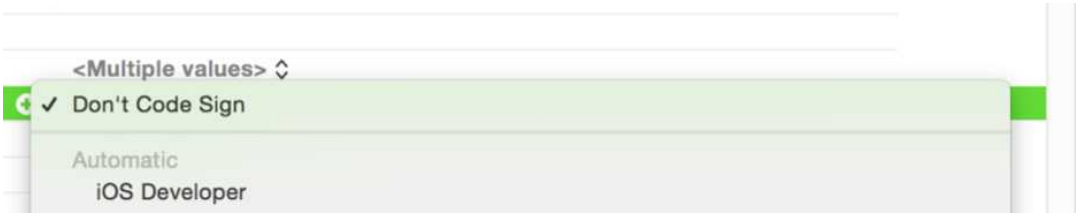


3.1. Dans cette fenêtre, défilez jusqu'à la section intitulée ***Code Signing.***

Dans cette section, vous retrouverez une sous-catégorie nommée **Code Signing Identity.**

3.2. Changez les paramètres suivants :

Code Signing
3.2.1. ***Code Signing Identity*** **: Automatic | iOS Developer**



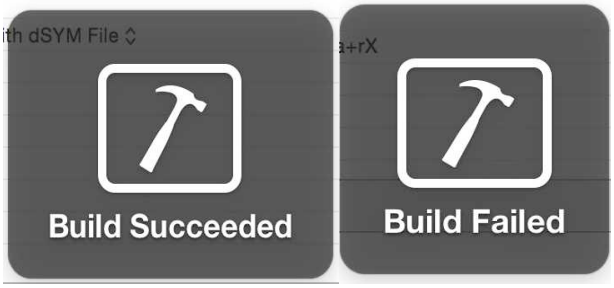
Votre signature numérique sera désormais utilisée pour compiler l'application.

Sauvegardez votre projet (⌘+S)

Il ne vous reste qu'à compiler et envoyer votre app sur l'appareil.

4. Appuyez sur le bouton « Play » dans l'entête de la fenêtre du projet.

Votre projet compilera les applications. Si tout se passe comme prévu, vous recevrez le message **Build Succeded.** Si une ou plusieurs erreurs sont présentes, vous verrez **Build Failed**. Consultez alors les messages d'erreurs et réglez les problèmes en conséquence. Sinon, réexportez votre projet depuis Unity®, en vous assurant de ne pas avoir omis une étape.



Si tout s'est bien déroulé, votre jeu s'exécutera.

5. Vous pouvez alors débrancher le câble USB de votre Mac.

Notez-bien!

Notez qu'il est normal que le jeu se ferme sur votre portable. Relancez-le directement depuis votre iPhone ou iPad.

Cet ouvrage est maintenant complété. J'espère que vous l'avez apprécié et que vous allez réaliser de merveilleux projets avec les connaissances acquises. Je vous souhaite donc :

Bon développement !

unity

Glossaire

#

&& ..... Voir conditions
++ ..... Voir opérations
>= ..... Voir conditions
|| ..... Voir conditions
2D Project 167
3D Gizmos 35

A

Add Branch Group 214
Add Motion Field 336
Add Terrain Texture 205
Add-Grass Texture 224
Albedo 22
Alpha Blended Premultiply 257
Ambient GI 230
Ambient Intensity 230
Ambient Source 230
Android 423
Animation 300
Animation sheet 252
Animation View 39
Animator 40, 264, 268, 300, 302, 334
Animator Controller 264, 331
APK 430
Apple iOS 432
Application.LoadLevel() 131, 296
Array ..... Voir variables
Aspect ratio 269
Asset Store 26, 395
Asset Store Preview 229
Assets 21
Assets Store 228
Audio Listener 136
Audio Mixer 40
Audio Source 135
audio.Play 296
AudioListener 22
AudioSource 22, 293, 390, 392, 394
Automate Thresholds 336
Awake() 104
Axes 315
AZERTY ..... Voir Keys

B

Bake Light Probes For Trees 222
Bending 223
Billboard 224
Billboard Start 222
Blend tree 332
Blending 337
Bloom and Glow 236
boolean ..... Voir variables
Border 168
Boucles 99
Branch Material 210, 214
Break Chance 212
Break Location 212
Break Material 210
Build 51
Build And Run 52
Build Settings 127, 138, 174, 425, 444
Button 169, 177

C

Cache compression ..... Voir GI Cache
Cache Folder Location ..... Voir GI Cache
Cache Server 62
Cache size is ..... Voir GI Cache
Camera 272
Cameras 50
Canvas 146, 169, 176, 273, 284, 288, 364
Canvas Scaler ..... Voir Canvas
Cap Smoothing 212
Capsule Collider 220, 352
Certificates 437
Classes 108
Clean Cache ..... Voir GI Cache
Clips 336
Code Signing Identity 450
Collect Detail Patches 222
Collider 21
Colors 60
Component 21
Components 36
Conditions 95, 303, 338
Console 39
Constructeur ..... Voir classes
Continuous Baking 298
Create Empty 299
Create State 302
Create Wind Zone 213
Crinkliness 212
Cross-Platform Settings 427, 446
Cube 66, 353
Cursor.lockState 390
Cursor.visible 390

Custom cache location ..... Voir GI Cache

D

Destroy() 359
Detail Density 222
Detail Distance 222
Detail Texture 224
Development Build 52
Diffuse ..... Voir Albedo
Directional Light 67, 377
Distribution 209
do… while ..... Voir boucles
Draw 222
Dry Color 224

E

Edge Turbulence 213
else ..... Voir conditions
else if ..... Voir conditions
Emission 23
Environment Lighting 230, 379
EventSystem 152
Example Project 17
Exigences minimales d'Unity 3
extends ..... Voir héritage
External Tools 59

F

Fade Length 222
Favorites 26
fillAmount 156, 159, 160
filtre 26
FirstPersonCharacter 233
FixedUpdate() 104
Flare 232
Flare Layer 233
float ..... Voir variables
fonctions 102
for ..... 314, 320, Voir boucles
Freemium 414
Freeze Position 346, 362
Freeze Rotation 346
Frequency 209
From New Blend Tree 335
Frond Count 213
Frond Crease 213
Frond Material 211
Frond Range 213
Frond Rotation 213
Frond Width 213
Fronds 212
function ..... Voir fonctions

G

Game View 33
GameObject 21
General 58
Geometry 210, 218
Geometry Mode 210, 218
GetCurrentAnimatorStateInfo() 351
GI Cache 61
Gizmo 29, 32, 269, 274
Gizmos 35
Grass Tint 223
Group Seed 209
Growth Angle 209
Growth Scale 209

H

Halo 125, 346
Hard Shadows ..... Voir Shadow Type
Healthy Color 224
Height 199
Heightmap Resolution 199
Héritage 110
Horizontal Align 218

I

Icon 35
Icons 38
if ..... Voir conditions
Image 154, 285, 289, 290, 291
ImageFilled 157
Images Effects 234
in ..... Voir conditions
Input 315
Input Manager 115
Input.GetAxis("Mouse #") 115, 349
Input.GetButtonUp 317
Input.GetKey(Key) 133
Inspector 36
Instantiate() 182
int ..... Voir variables
Invoke 191
IP Adress ..... Voir Cache Server
IsName() 352, 386

K

Keys 61

L

L'outil des branches 209
Labels 38
LateUpdate() 104
layer 278, 281, 337
Layout 42
Length 212
Light Probe Group 374
Lighting 40, 230, 298, 373, 379
lightmaps 128

Lights....51
LOD Multiplier....210
Loop Time....301

M

magnitude....306
Main Turbulence....213
Main Wind....213
Mask....337
Material....22, 218
Max Height....224
Max Mesh Trees....223
Max Width....224
Maximum Cache Size (GB)....Voir GI Cache
Mesh....21, 218
Methods....109
Min Height....224
Min Width....224
MonoBehaviour....105
Motion Blur....294
Motion fields....336
Multiply....260

N

New State....303
Next Frame....33
Noise....212
Noise Scale U....212
Noise Scale V....212
Noise Spread....224
Normal Map....23

O

Offline Web Player....54
OnApplicationFocus()....105
OnApplicationPause()....105
OnApplicationQuit()....105
OnBecameInvisible()....105
OnBecameVisible()....105
OnCollisionEnter()....104
OnCollisionEnter2D()....Voir OnCollisionEnter()
OnCollisionExit()....104
OnCollisionExit2D()....Voir OnCollisionExit()
OnCollisionStay()....104
OnCollisionStay2D()....Voir OnCollisionStay()
OnDestroy()....105
OnDisable()....105
OnEnable()....105
OnGUI()....104
OnTriggerEnter()....105
OnTriggerEnter2D()....Voir OnTriggerEnter()
OnTriggerExit()....105
OnTriggerExit2D()....Voir OnTriggerExit()
OnTriggerStay()....105
OnTriggerStay2D()....Voir OnTriggerStay()
Opérations....91
Optimization....448
Orbite....30, 31
Other Settings....428, 447
Outline....81, 171

P

Paint height....200, 202
Paint texture....204, 248
Panel....176
Parameters....302, 332
Particle System....124, 253, 257, 261, 339, 341, 342, 343
Particle System Destroyer....396
Pause....33
Perpendicular Align....218
Per-Platform Settings....427
Physic Material....72
PhysX....64
Play()....33, 127, 306, 359
Player Settings....138
Point light....250
Prefab....21, 49
Prefabs....37
Preferences....58
Probes....375
Profiler....39
Project browser....24
pseudo-code....84

R

Radius....212
Raise / Lower Terrain....200
Random.Range....120, 357
Rect tool....32
Reflection Bounces....230
Reflection Intensity....230
Relative Length....212
relativeVelocity....306
Renderer....21
Resolution and Presentation....427, 446
return....359, Voir fonctions
rig generic....332
Rigidbody....22, 72, 362
RigidBodyFPSController....233

S

SampleScenes....Voir Example Project
Save Layout...Voir Layout
Scene View....29
Screen Match Mode....289
Screen Space Ambient Occlusion....237
SDK Manager....424
Seek Sun....212
SendMessage()....185, 371, 384
Set Transition....304

SetActive() 131, 314, 371, 387
SetFloat 349
Shader 22
Shader: Standard 70
Shadow 148, 170, 178, 307
Shadow Type 67
Shape 211, 218
Size 218, 223
Skin Mesh Renderer 374
Skybox 230
Smooth Height 201, 204
Soft Shadows ..... Voir Shadow Type
Specular 22
Speed 223
Sphere 66, 295, 345
Sphere Collider 295
Spotlight 345
Spread Bottom 212
Spread Top 212
Standalone 54
Standard Assets 395
Start() 104
Streamed 53
String ..... Voir variables
Sun 230
Sun Shafts 235
super ..... Voir héritage
Surdéfinition 105
Switch Platform 52

T

Tag 117, 295, 296, 355, 359, 371
Tags 354
Tags & Layers 79, 117, 271, 354
Terrain 199
Terrain settings 199
Text 147, 383
Texture 22
Texture Sheet Animation 254
Threshold ..... Voir Thresholds
Thresholds 336
Time.deltaTime 115
Time.timeScale 318
ToString("D2") 152
Transform Tools 32
transform.Translate 115
Translate 350
translation 31
Tree 208
Tree & Detail Objects 222
Tree Distance 222

U

UIButton 309, 311, 367
UIText 307, 365, 366
Unity Remote 417
Update() 104
Use Cache Server ..... Voir Cache Server

V

Variables 86
velocity 356
Views 23, 39
Vignette And Chromatic Aberration 239

W

WaitForSeconds 296, 386
Water4Advanced 226
WaterBasicNightime 244
Weight 337
Weld Lenght 212
while ..... Voir boucles
Wind 213, 218
Wind Settings 223
Window 42

X

Xcode SDK 433

Y

yield 296, 386

Z

zoomer 31